TURBO PASCAL®
für Windows

WINDOWS-
REFERENZ-
HANDBUCH

BORLAND

Turbo Pascal für Windows

Windows-Referenzhandbuch

BORLAND

Copyright © 1991 by Borland International, INC.
Jede Vervielfältigung dieses Handbuches sowie des Softwareprogrammes Turbo Pascal für Windows wird strafrechtlich verfolgt. Die Rechte an der Dokumentation und die Rechte an dem Softwareprogramm Turbo Pascal für Windows liegen bei Borland International.

Der rechtmäßige Erwerb der Programmdisketten und der Handbücher erlaubt die Nutzung der Programme analog der Benutzung eines Buches. Entsprechend der Unmöglichkeit, daß ein Buch zugleich an verschiedenen Orten von mehreren Personen gelesen wird, darf das Softwareprogramm Turbo Pascal für Windows nicht gleichzeitig von verschiedenen Personen an verschiedenen Orten und auf verschiedenen Geräten benutzt werden. Diskettenkopien dürfen lediglich zum Zweck der Datensicherung angefertigt werden.

Sie (der Lizenznehmer von Turbo Pascal für Windows) dürfen eigene Programme, die Sie mit dem Sprachcompiler Turbo Pascal für Windows geschrieben und compiliert haben, ohne Einschränkung oder Verpflichtung zu zusätzlichen Lizenzgebühren verwenden, weitergeben oder verkaufen, solange diese Programme einen Copyright-Vermerk enthalten. Unter Copyright-Vermerk verstehen wir entweder Ihren eigenen Copyright-Vermerk oder gegebenenfalls einen Hinweis wie »Created using © Borland International 1991«.

Die Beispielprogramme von Turbo Pascal für Windows, die auf den Originaldisketten mitgeliefert werden demonstrieren die verschiedenen Eigenschaften von Turbo Pascal für Windows. Sie dienen ausschließlich zu Lehrzwecken. Sie dürfen diese Beispielprogramme editieren oder für den eigenen Gebrauch modifizieren. Sie dürfen diese Programme jedoch nicht als Ganzes oder in Teilen in compilierter Form oder als Quelltext weitergeben oder verkaufen. Sie dürfen die Routinen aus den Beispielprogrammen in eigenen Anwendungsprogrammen verwenden, solange die resultierenden Programme nicht einem Bespielprogramm oder einem Teil eines Beispielprogrammes in Erscheinung oder Funktion gleichen.

Einschränkung der Gewährleistung
Es wird keine Garantie für die Richtigkeit des Inhaltes dieses Handbuches übernommen. Da sich Fehler, trotz, aller Bemühungen, nie vollständig vermeiden lassen, sind wir für Hinweise jederzeit dankbar.

Herausgeber
Borland GmbH
Lindwurmstraße 88
Postfach 15 03 40
8000 München 2
Telefon 0 89 / 72 01 00
Telefax 0 89 / 77 93 39
Telex 5 21 26 37 mcm d

Ubertragung ins Deutsche:
Marcus O. Lerch, Arne Schäpers und Borland GmbH

Typographische Konzeption des Innenteils: Rudolf Paulus Gorbach

Grafiken: Sid Gastl, Andreas Höhne

PostScript-Belichtung: CD Computer & Dokumentations GmbH, Aalen

Druck und Bindung: Schoder Druck GmbH, Gersthofen

Papier: Schleipen GmbH, Bad Dürkheim. Das verwendete Papier ist säurefrei und alterungsbeständig. Die Produktion erfolgt in einem geschlossenen Wasserkreislauf ohne Einsatz von optischen Aufhellern und unter Verwendung von chlorarm gebleichten, holzfreien Zellstoffqualitäten. Die Abwasserreinigung wird in einer betriebseigenen chemisch-biologischen Kläranlage vorgenommen.

April 1991

Inhalt

Einführung 1

Teil I Windows-API-Referenz 3

1. Windows-Stile und -Konstanten 5

2. Windows-Funktionen 79

3. Windows-Botschaften 449

4. Windows-Typen 545

Teil II ObjectWindows-Referenz 575

5. Verzeichnis der Objekte 577

6. Außerhalb der Objekte 653

Index 667

Einführung

Dieses Handbuch dokumentiert in Teil I (Kapitel 1 bis 4) alle Eigenschaften von Microsoft-Windows – einschließlich aller Konstanten, Stile, Botschaften und API-Funktionen der Pascal-Schnittstelle zu Windows – deren Kenntnis für den Programmierer notwendig ist.

Des weiteren finden Sie in Teil II (Kapitel 5 bis 6) eine Erläuterung der Objekte, Datentypen und globalen Variablen der ObjectWindows-Bibliothek.

Kapitel 1 »Windows-Stile und -Konstanten« beschreibt die von der Windows-API benutzten Konstanten und Stile, die in der Turbo-Pascal-Unit *WinTypes* definiert sind.

Kapitel 2 »Windows-Funktionen« enthält in alphabetischer Reihenfolge eine Auflistung aller Funktionen und Prozeduren der Windows-API, die in der Turbo-Pascal-Unit *WinProcs* definiert sind.

Kapitel 3 »Windows-Botschaften« enthält eine Auflistung und Erläuterung aller Botschaften, die für die Kommunikation zwischen Windows und Anwendungen benötigt werden.

Kapitel 4 »Windows-Typen« beschreibt alle einfachen und strukturierten Typen der Windows-Schnittstelle.

Kapitel 5 »Verzeichnis der Objekte« enthält eine alphabetische Auflistung aller Objekte der ObjectWindows-Bibliothek mit der detaillierten Beschreibung der Felder und Methoden jedes Objekttyps.

Kapitel 6 »Außerhalb der Objekte« erläutert Datentypen, Konstanten und Routinen der Bibliothek, die nicht zur Objekthierarchie von ObjectWindows gehören.

Die Typographie in den Handbüchern

Zur Verdeutlichung von Zusammenhängen verwenden wir in diesem Handbuch mehrere unterschiedliche Schriftarten:

`Schreibmaschine` steht für Syntaxdefinitionen, Eingaben des Benutzers, Programmlistings und Ausgaben, die von Beispielprogrammen erzeugt werden.

Fettschrift kennzeichnet reservierte Wörter von Turbo Pascal für Windows wie **nil**, **and** oder **or**.

Kursiv wird für Funktionsnamen (wie *AccessResource*) sowie für Namen von Variablen und Bezeichnern im laufenden Text benutzt.

[Esc] für die Tasten Ihrer Tastatur, wie beispielsweise in »Ein Druck auf [Esc] entfernt das Menü wieder vom Bildschirm«.

I. Windows - API-Referenz

Teil I dieses Handbuches enthält Informationen über die Anwendungsprogrammierungsschnittstelle (API) von Microsoft Windows.

Die API besteht aus Datenstrukturen, Funktionen und Botschaften, die das Gerüst Windows-Schnittstelle darstellen. Turbo-Pascal-Entwickler haben vollen Zugriff auf die Windows-Funktionen, auch wenn sie die Funktionen nur dann einzusetzen brauchen, wenn die von ObjectWindows unterstützten Objekte nicht ausreichen.

Die Definitionen der Datenstrukturen (beispielsweise *TRect* und *TWndClass*) finden sich in der Datei WINTYPES.PAS. In diesem Handbuch finden Sie häufig Hinweise auf Windows-Konstanten wie *ws_Border* oder *sw_ShowWindow*, die ebenfalls in WINTYPES.PAS definiert und in Kapitel 1 »Windows-Stile und -Konstanten« beschrieben sind. Diese Konstanten werden auch als Argumente in Aufrufen von ObjectWindows-Methoden verwendet.

Kapitel 2 »Windows-Funktionen« enthält alle verfügbaren Windows-Funktionen und Kapitel 3 »Windows-Botschaften« alle verfügbaren Windows-Botschaften in alphabetischer Reihenfolge.

Die Verwendung der Windows-API-Funktionen, -Botschaften und -Stile wird im *Windows-Programmierhandbuch*, insbesondere in Kapitel 7 »Überblick über ObjectWindows«, ausführlich erläutert.

1. Windows-Stile und -Konstanten

Dieses Kapitel erläutert die häufig verwendeten Konstanten für den Aufruf von Windows-Funktionen und den Empfang von Windows-Botschaften (siehe Kapitel 2 »Windows-Funktionen« und Kapitel 3 »Windows-Botschaften«). Die Konstanten sind in gebräuchlichen Gruppen alphabetisch angeordnet. Soweit möglich, wurden sie über ihre Präfixe angeordnet und sortiert. So finden sich alle Konstanten, deren Name mit *bi_* beginnt, in der Gruppe »*bi_Xxxx*«. Konstanten, die keine Präfixe besitzen, wurden nach ihrer Funktion angeordnet. Die Konstanten *ComplexRegion*, *NullRegion* und *SimpleRegion* bestimmen Typen grafischer Regionen und werden als Parameter für dieselben Funktionen verwendet. Sie wurden deshalb in der Gruppe »Region-Flags« angeordnet.

bi_Xxxx

Diese Konstanten werden als Werte für das Feld *biCompression* der Struktur *TBitmapInfoHeader* übergeben, wenn die Funktion *CreateDIBitmap* aufgerufen wird.

bi_RGB	Bitmap wird nicht komprimiert.
bi_RLE8	Bitmap verwendet ein Format mit 8 Bit pro Pixel.
bi_RLE4	Bitmap verwendet ein Format mit 4 Bit pro Pixel.

bn_Xxxx

Die folgenden Benachrichtigungscodes beziehen sich auf Aktionschalter. Sie werden vom jeweiligen Schalter in einer *wm_Command*-Botschaft übergeben und zeigen eine Aktion des Anwenders an.

bn_Clicked	Zeigt dem besitzenden Fenster an, daß ein Aktionsschalter betätigt wurde.
bn_DoubleClicked	Zeigt dem besitzenden Fenster an, daß der Anwender einen Doppelklick auf einem besitzergezeichneten Aktionsschalter oder einem Schaltfeld durchgeführt hat. Dieser Code bezieht sich nur auf Schalter des Stils *bs_OwnerDraw* und auf Aktionsschalter.

bs_Xxxx

Diese Konstanten bestimmen den Stil des virtuellen Pinsels. Sie werden im Feld *lbStyle* des Record *TLogBrush* durch die Funktion *CreateBrushIndirect* übergeben.

bs_DIBPattern	Legt ein Pinselmuster fest, das durch eine geräteunabhängige Bitmap-Beschreibung (DIB) definiert ist.
bs_Hatched	Legt einen Schraffurpinsel fest.
bs_Hollow	Legt einen Pinsel fest, der mit der Farbe des Hintergrunds malt und dadurch löscht.
bs_Null	Wie *bs_Hollow*.
bs_Pattern	Legt einen gemusterten Pinsel fest, der durch ein Speicher-Bitmap definiert ist.
bs_Solid	Legt einen kompakten Pinsel fest.

Diese Konstanten dienen zur Festlegung der Schalterstile. Sie werden verwendet, wenn Schalter mit *CreateWindow* und *CreateWindowEx* erzeugt werden.

bs_3State	Dieser Stil erlaubt die Darstellung des Schalters im ausgewählten, nichtausgewählten und im nichtaktivierbaren Zustand. Der nichtaktivierbare Zustand wird üblicherweise verwendet, um zu zeigen, daß ein Auswahlfeld deaktiviert wurde.
bs_Auto3State	Dieser Schalterstil ist identisch mit *bs_3State*. Allerdings ändert der Schalter seinen Zustand automatisch, wenn der Benutzer ihn anklickt.
bs_AutoCheckBox	Identisch mit *bs_CheckBox*, außer, daß das Auswahlfeld automatisch seinen Zustand ändert, wenn es der Benutzer anklickt.
bs_AutoRadioButton	Identisch mit *bs_RadioButton*, außer, daß das Schaltfeld ausgewählt dargestellt wird, wenn es der Anwender anklickt. Die Anwendung wird mit der Botschaft *bn_Clicked* verständigt und die Auswahlmarkierungen werden von allen anderen Schaltfeldern in der Gruppe entfernt.
bs_CheckBox	Bestimmt ein kleines rechteckiges Auswahlfeld, das angewählt werden kann; seine Umrandung wird fett dargestellt, wenn der Benutzer ihn anklickt. Zugehöriger Text erscheint rechts vom Schalter.
bs_DefPushButton	Dieser Schalterstil verhält sich ähnlich wie *bs_PushButton*. Allerdings wird dieser Aktionsschalter mit einer fetten Umrandung dargestellt. Er ist die voreingestellte Antwort des Benutzers, solange keine andere Auswahl mit Maus oder Tastatur getroffen wird. Text

wird innerhalb des Schalters angezeigt. Windows sendet eine Botschaft an das übergeordnete Fenster, wenn der Benutzer den Schalter anklickt.

bs_GroupBox Bestimmt ein Reckteck, in dem andere Schalt- oder Auswahlfelder angeordnet werden. Text wird in der oberen linken Ecke des Rechtecks angezeigt.

bs_LeftText Bewirkt, daß Text auf der linken Seite des Schaltfeldes oder Auswahlfeldes erscheint. Verwenden Sie diesen Stil mit den Stilen *bs_CheckBox*, *bs_RadioButton* oder *bs_3State*.

bs_OwnerDraw Bestimmt einen besitzergezeichneten Schalter. Das übergeordnete Fenster wird benachrichtigt, wenn der Schalter angeklickt wird. Zusätzlich zu den mit der Botschaft *wm_Command* übergebenen normalen Benachrichtigungen erhält es Aufforderungen zum Zeichnen, Invertieren oder Deaktivieren des Schalters.

bs_PushButton Bestimmt einen Aktionsschalter. Text wird innerhalb des Schalters dargestellt. Das Steuerelement sendet eine Botschaft immer dann an sein übergeordnetes Fenster, wenn der Benutzer den Schalter anklickt.

bs_RadioButton Bestimmt ein kleines kreisförmiges Schaltfeld, das angewählt werden kann; seine Umrandung wird fett dargestellt, wenn der Benutzer ihn anklickt. Text erscheint rechts vom Schaltfeld. Üblicherweise werden zwei oder mehr Schaltfelder zusammengruppiert, um sich gegenseitig ausschließende Auswahlen darzustellen. Es kann nicht mehr als ein Schaltfeld in der Gruppe zu jeder Zeit ausgewählt sein.

cb_Xxxx

Diese Werte werden von Windows-Kombinationsfensterbotschaften (zum Beispiel *cb_AddString*) übergeben. Ein negativer Wert zeigt einen Fehler an.

cb_Err Es ist ein Fehler aufgetreten. Die Aktion konnte nicht durchgeführt werden.

cb_ErrSpace Der Platz im Kombinationsfenster reicht zur Durchführung der Aktion nicht aus.

cb_Okay Kein Fehler.

cbm_Init

Die Übergabe von *cbm_Init* als Parameter *dwUsage* der Funktion *CreateDIBitmap* zeigt an, daß ein Speicher-Bitmap initialisiert werden soll.

cbn_Xxxx

Die folgenden Benachrichtigungscodes werden in den *wm_Command*-Botschaften von Kombinationsfenstern übergeben. Sie zeigen an, daß und welche Aktion stattgefunden hat.

cbn_DblClk Wird an das übergeordnete Fenster übergeben, wenn der Anwender einen String in der Liste des Kombinationsfensters durch einen Doppelklick mit der Maus auswählt.

cbn_DropDown Informiert das übergeordnete Fenster eines Kombinationsfensters über die Aktivierung einer Drop-Down-Liste.

cbn_EditChange Benachrichtigt das übergeordnete Fenster eines Kombinationsfensters, daß der Anwender den Text im Editierfeld des Kombinationsfensters verändert hat.

cbn_EditUpdate Benachrichtigt das übergeordnete Fenster eines Kombinationsfensters, daß der Text im Editierfeld verändert wurde und neu dargestellt werden muß.

cbn_ErrSpace Benachrichtigt das übergeordnete Fenster eines Kombinationsfensters, wenn im System kein Speicher mehr verfügbar ist.

cbn_KillFocus Benachrichtigt das übergeordnete Fenster eines Kombinationsfensters, daß das Kombinationsfenster den Fokus verloren hat.

cbn_SelChange Benachrichtigt das übergeordnete Fenster eines Kombinationsfensters, daß die Auswahl in der Liste des Kombinationsfensters geändert wurde.

cbn_SetFocus Benachrichtigt das übergeordnete Fenster eines Kombinationsfensters, daß das Kombinationsfenster den Fokus erhalten hat.

cbs_Xxxx

Diese Konstanten werden zur Festlegung der Kombinationsfensterstile verwendet, wenn Kombinationsfenster mit den Funktionen *CreateWindow* und *CreateWindowEx* erzeugt werden.

cbs_AutoHScroll Dieser Kombinationsfensterstil rollt den Text im Editierfeld automatisch nach links, wenn der Benutzer ein Zeichen am Ende der Zeile eintippt. Wenn dieser Stil nicht gesetzt ist, ist nur Text erlaubt, der in die rechteckige Umrandung paßt.

cbs_DropDown Ähnlich wie *cbs_Simple*, außer, daß die Liste nur dann

angezeigt wird, wenn der Benutzer ein Symbol neben dem Auswahlfeld auswählt.

cbs_DropDownList Ähnlich wie *cbs_DropDown*, außer, daß das Editierfeld durch ein statisches Textelement ersetzt wird, das die momentane Auswahl aus der Liste anzeigt.

cbs_HasStrings Dieser Stil kann in Verbindung mit *cbs_OwnerDrawFixed* oder *cbs_OwnerDrawVariable* verwendet werden. Ein besitzergezeichnetes Kombinationsfenster enthält Einträge, die aus Strings bestehen. Das Kombinationsfenster behält den Speicher und die Zeiger für die Strings bei, so daß die Anwendung die Botschaft *lb_GetText* verwenden kann, um den Text für einen bestimmten Eintrag abzurufen.

cbs_NoIntegralHeight Dieser Kombinationsfensterstil entspricht der Größe des Kombinationsfensters bei seiner Erzeugung.
Normalerweise wird diese Größe den Einträgen im Kombinationsfenster so angepaßt, daß alle Einträge komplett dargestellt werden können.

cbs_OEMConvert Dieser Stil kann in Verwendung mit *cbs_Simple* oder *cbs_Dropdown* verwendet werden. Text, der in das Editierfeld eines Kombinationsfensters eingegeben wird, wird vom ANSI-Zeichensatz in den OEM-Zeichensatz und dann zurück in ANSI umgewandelt. Dies sorgt für eine exakte Zeichenumwandlung, wenn die Anwendung die Funktion *AnsiToOem* aufruft, um einen ANSI-String im Kombinationsfenster in OEM-Zeichen umzuwandeln.
Dieser Stil ist für Kombinationsfenster hilfreich, die Dateinamen enthalten und wird nur für Kombinationsfenster verwendet, die mit den Stilen *cbs_Simple* oder *cbs_DropDown* erzeugt wurden.

cbs_OwnerDrawFixed Der Besitzer des Kombinationsfenster ist für die Darstellung der Liste zuständig. Die Einträge in der Liste haben alle die gleiche Höhe.

cbs_OwnerDrawVariable Der Besitzer des Kombinationsfensters ist für die Darstellung der Liste zuständig. Die Einträge in der Liste haben eine variable Höhe.

cbs_Simple Die Liste des Kombinationsfensters wird zu jeder Zeit angezeigt. Die aktuelle Auswahl aus der Liste wird im Editierfeld angezeigt.

cbs_Sort Kombinationsfenster dieses Stils besitzen eine sortierte Liste. Strings, die in die Liste eingegeben werden, werden automatisch sortiert. Die Sortierreihenfolge kann für Kombinationsfenster der Stile *cbs_Owner-*

DrawFixed oder *cbs_OwnerDrawVariable* voneinander abweichen.

cc_Xxxx

Diese Konstanten repräsentieren die Bogenzeichnungsfähigkeiten eines Geräts.

cc_None	Keine Bogenzeichnungsfähigkeit.
cc_Chord	Kreisbögen können dargestellt werden.
cc_Circles	Kreise können dargestellt werden.
cc_Ellipses	Ellipsen können dargestellt werden.
cc_Interiors	Ausfüllungen können dargestellt werden.
cc_Pie	Tortendiagramme können dargestellt werden.
cc_Styled	Verschiedene Linien können dargestellt werden.
cc_Wide	Breite Linien können dargestellt werden.
cc_WideStyled	Verschiedene breite Linien können dargestellt werden.

ce_Xxxx

Diese Flags bestimmen einen Schnittstellenfehler und werden von der Funktion *GetCommError* zurückgegeben.

ce_Break	Die Hardware stellt eine Abbruchbedingung fest.
ce_CTSTO	Clear-to-send-Unterbrechung (time out). CTS ist für die durch das Feld *CtsTimeout* der *TDCB*-Struktur festgelegte Dauer herabgesetzt, während gleichzeitig versucht wird, ein Zeichen zu übertragen.
ce_DNS	Die parallele Schnittstelle ist nicht im Zustand »select«.
ce_DSRTO	Data-set-ready-Unterbrechung. DSR ist für die durch das Feld *DsrTimeout* der *TDCB*-Struktur festgelegte Dauer herabgesetzt, während gleichzeitig versucht wird, ein Zeichen zu übertragen.
ce_Frame	Die Hardware stellt einen Framing-Fehler fest.
ce_IOE	Ein I/O-Fehler tritt während des Versuchs der Kommunikation mit einer parallelen Schnittstelle auf.
ce_Mode	Der gewünschte Modus wird nicht unterstützt, oder der Parameter *Cid* ist ungültig. Ist der Parameter gesetzt, ist dies der einzig mögliche Fehler.

ce_OOP Der Drucker an einer parallelen Schnittstelle meldet Papierende.

ce_Overrun Ein Zeichen wird von der Hardware nicht rechtzeitig gelesen und vom folgenden Zeichen überschrieben. Das Zeichen ist verloren.

ce_PTO Bei der Kommunikation mit der parallelen Schnittstelle tritt eine Unterbrechung auf.

ce_RLSDTO Receive-line-signal-detect-Unterbrechung. RLSD ist für die durch das Feld *RlsdTimeout* der *TDCB*-Struktur festgelegte Dauer herabgesetzt, während gleichzeitig versucht wird, ein Zeichen zu übertragen.

ce_RXOver Überlauf der Empfangs-Warteschlange. Der Platz in der Empfangs-Warteschlange reicht nicht aus oder die Übertragung wurde nach Erhalt des Eof-Zeichens fortgesetzt.

ce_RXParity Die Hardware stellt einen Parity-Fehler fest.

ce_TXFull Die Übertragungs-Warteschlange ist voll. Gleichzeitig wird versucht, ein Zeichen in die Übertragungs-Warteschlange zu schreiben.

cf_Xxxx

Diese Konstanten bezeichnen Formate der Zwischenablage, die verwendet werden, wenn Daten an die Zwischenablage übergeben oder aus ihr entnommen werden. Außerdem werden sie im DDE-Protokoll (dynamischer Datenaustausch) verwendet. Die Konstanten werden mit den folgenden Funktionen verwendet: *EnumClipboardFormats*, *GetClipboardData*, *GetClipboardFormatName*, *GetPriorityClipboardFormat*, *IsClipboardFormatAvailable* und *SetClipboardData*.

cf_Bitmap Ein Handle zu einem Windows-Bitmap.

cf_DIB Ein Speicherblock, der eine geräteunabhängige Bitmap-Beschreibung enthält.

cf_DIF Software Arts' Data Interchange Format.

cf_DSPBitmap Bitmap-Anzeigeformat, das einem privaten Format zugeordnet ist.

cf_DSPMetaFilePict Metafile-Darstellungsformat, das einem privaten Format zugeordnet ist.

cf_DSPText Textdarstellungsformat, das einem privaten Format zugeordnet ist.

cf_MetaFilePict Metafile-Bildformat in einer *MetaFilePict*-Datenstruktur

cf_OEMText — Textformat, das Zeichen aus dem OEM-Zeichensatz enthält. Jede Zeile endet mit einer Kombination Wagenrücklauf/Zeilenschaltung (CR-LF). Ein Null-Zeichen kennzeichnet das Datenende.

cf_OwnerDisplay — Besitzeranzeigeformat. Der Besitzer der Zwischenablage muß das Anwendungsfenster der Zwischenablage anzeigen und aktualisieren und empfängt die Botschaften *wm_AsKCBFormatName*, *wm_HScroll-Clipboard*, *wm_PaintClipboard*, *wm_SizeClipboard* und *wm_VScrollClipboard*.

cf_Palette — Handle zu einer Farbpalette. Immer, wenn eine Anwendung Daten in der Zwischenablage speichert, die von einer Farbpalette abhängen oder eine solche voraussetzen, dann sollte sie die Palette ebenfalls in der Zwischenablage speichern.
Wenn die Zwischenablage Daten im Format *cf_Palette* (virtuelle Farbpalette) enthält, sollte die Anwendung davon ausgehen, daß andere Daten in der Zwischenablage unabhängig von dieser virtuellen Palette realisiert sind. Die Zwischenablage-Viewer-Anwendung (CLIPBRD.EXE) benutzt als aktuelle Palette ein Objekt im Format *cf_Palette* in der Zwischenablage, wenn die anderen Formate in der Zwischenablage dargestellt werden.

cf_PrivateFirst bis *cf_PrivateLast* — Bereich von Integerwerten, die für private Formate benutzt werden. Daten-Handles, die Formaten in diesem Bereich zugeordnet sind, werden nicht automatisch freigegeben.
Alle Daten-Handles müssen von der Anwendung freigegeben werden, bevor die Anwendung beendet oder wenn eine Botschaft *wm_DestroyClipboard* empfangen wird.

cf_SYLK — Microsoft-Symbolic-Link (SYLK) Format.

cf_Text — Textformat. Jede Zeile endet mit einer Kombination Wagenrücklauf/Zeilenschaltung (CR-LF). Ein Null-Zeichen kennzeichnet das Datenende.

cf_TIFF — Tag Image File Format.

cchDeviceName

Die Konstante *cchDeviceName* begrenzt die Länge eines Gerätenamensstrings auf 32 Zeichen.

clip_Xxxx

Diese Konstanten bestimmen die Clipping-Präzision von Schriften, die mit der Funktion *CreateFont* erzeugt werden. Sie steuern die Textdarstellung, wenn der Text teilweise außerhalb der Clipping-Region liegt.

clip_Character_Precis
clip_Default_Precis
clip_Stroke_Precis

color_Xxxx

Diese Codes legen Eigenschaften der Windows-Anwenderschnittstelle fest. Programmierer können deren Farben direkt mit der Funktion *SetSysColor* setzen.

color_ActiveBorder	Umrahmung des aktiven Fensters.
color_ActiveCaption	Überschrift des aktiven Fensters.
color_AppWorkSpace	Hintergrundfarbe von Multidokumentschnittstellenrahmenfenstern (MDI).
color_Background	Windows-Desktop.
color_BtnFace	Flächenschattierung von Aktionsschaltern.
color_BtnShadow	Rahmenschattierung von Aktionsschaltern.
color_BtnText	Text in Aktionsschaltern.
color_CaptionText	Text in Überschrift, Bildgrößenfeld und Bildlaufpfeilfeld.
color_GrayText	Grauer (nichtaktivierbarer) Text. Diese Farbe wird auf 0 gesetzt, wenn der momentane Bildschirmtreiber einen echten Grauton nicht unterstützt.
color_HighLight	Kennzeichnet die gewählte Option in einem Steuerelement.
color_HighLightText	Text einer gewählten Option in einem Steuerelement.
color_InactiveBorder	Umrahmung eines deaktivierten Fensters.
color_InactiveCapiton	Überschrift eines deaktivierten Fensters.
color_Menu	Zeilenmenühintergrund.
color_MenuText	Text in Zeilenmenüs.
color_ScrollBar	Hintergrund einer Bildlaufleiste.
color_Window	Fensterhintergrund.
color_WindowFrame	Fensterrahmen.
color_WindowText	Text in Fenstern.

com_Xxxx

Diese Flags können als Bit-Maske verwendet werden, um die Bitfeldeinstellungen im Feld *Flags* einer *TComStat*-Struktur abzurufen oder zu setzen.

Konstante	Abgerufenes Bit
com_CtsHold	*fCtsHold*
com_DsrHold	*fDsrHold*
com_RlsdHold	*fRlsdHold*
com_XoffHold	*fXoffHold*
com_XoffSent	*fXoffSent*
com_Eof	*fEof*
com_Txim	*fTxim*

cp_Xxxx

Diese Konstanten zeigen die Clipping-Fähigkeiten eines Geräts an.

cp_None Kein Clipping der Ausgabe möglich.

cp_Rectangle Clipping der Ausgabe in Rechtecke möglich.

cs_Xxxx

Diese Konstanten der Fensterklassenstile werden im Feld *style* der Datenstruktur *TWndClass* verwendet. Sie können mit dem Operator **or** verknüpft werden.

cs_ByteAlignClient Ausrichtung des Client-Bereichs eines Fensters auf eine Byte-Grenze (in der x-Richtung).

cs_ByteAlignWindow Ausrichtung eines Fenster auf eine Byte-Grenze (in der x-Richtung).

cs_ClassDC Gibt der Fensterklasse einen eigenen Gerätekontext, den sich ihre Instanzen teilen.

cs_DblClks Fenster erhält Doppelklickbotschaften.

cs_GlobalClass Bestimmt, daß die Fensterklasse eine globale Klasse der Anwendung ist. Die globale Klasse einer Anwendung wird von einer Anwendung oder Bibliothek erzeugt und ist für alle Anwendungen verfügbar.
Die Klasse wird zerstört, wenn die Anwendung oder Bibliothek, die die Klasse erzeugte, beendet wird. Es ist deshalb notwendig, daß alle Fenster, die mit der anwendungsglobalen Klasse erzeugt wurden, vor diesem Ereignis geschlossen werden.

cs_HRedraw Neuzeichnung des kompletten Fensters, wenn sich dessen Breite ändert.

cs_NoClose	Sperrt die Schließoption im Systemmenü des Fensters.
cs_OwnDC	Gibt jeder Fensterinstanz ihren eigenen Gerätekontext. Beachten Sie, daß der *cs_OwnDC*-Stil zwar verwendbar ist, aber für jeden Gerätekontext ungefähr 800 Byte Speicher benötigt werden.
cs_ParentDC	Das Fenster enthält den Gerätekontext des übergeordneten Fensters.
cs_SaveBits	Sichert einen Bildschirmbereich, der von einem Fenster verdeckt wird. Windows benutzt das gesicherte Bitmap, um den Bildschirmbereich wiederherzustellen, wenn ein Fenster bewegt wird. Windows stellt das Bitmap an seiner ursprünglichen Position dar und sendet keine *wm_Paint*-Botschaft an Fenster, die durch das Fenster verdeckt wurden, wenn der durch das Bitmap verwendete Speicher nicht freigegeben wurde und wenn andere Bildschirmaktionen das gespeicherte Bild nicht ungültig gemacht haben. Eine Anwendung sollte dieses Bit nur für kleine Fenster setzen, die nur kurz angezeigt und dann beseitigt werden, bevor andere Bildschirmaktivitäten stattfinden. Das Setzen dieses Bits für ein Fenster erhöht die Zeitdauer für die Darstellung des Fensters wegen der benötigten Zeit für die Speicherreservierung und Speicherung des Bitmap.
cs_VRedraw	Neuzeichnung des kompletten Fensters, wenn sich dessen Höhe ändert.

ctlcolor_Xxxx

Diese Flags zeigen den Typ des Steuerelements oder Dialogfensters an, das dargestellt werden soll. Sie werden mit der Botschaft *wm_CtlColor* übergeben.

ctlcolor_Btn	Schalter.
ctlcolor_Dlg	Dialogfenster.
ctlcolor_Edit	Editierfeld.
ctlcolor_ListBox	Liste.
cltcolor_Max	Vergrößerungsfeld.
ctlcolor_MsgBox	Meldungsfenster.
ctlcolor_ScrollBar	Bildlaufleiste.
ctlcolor_Static	Statisches Steuerelement.

cw_UseDefault

cw_UseDefault wird als Parameter mit den Funktionen *CreateWindow* oder *CreateWindowEx* verwendet. Sie veranlaßt Windows, dem neuerstellten Fenster eine Standardgröße bzw. -position zuzuweisen.

dc_Xxxx

Diese Konstanten werden bei Aufrufen der Funktion *GetDeviceCap* im Parameter *Index* verwendet, um die speziellen Fähigkeiten des Druckertreibers abzufragen. Das Argument *Index* kann folgende Werte annehmen:

Konstante	Abgefragte Fähigkeit
dc_Bins	Liste verfügbarer Schächte.
dc_Driver	Versionsnummer des Gerätetreibers.
dc_Duplex	Umfang der Duplex-Unterstützung.
dc_Extra	Die Anzahl Bytes, die am Ende eines *TDevMode*-Record für gerätespezifische Daten verwendet werden.
dc_Fields	Das Feld *dmFields* des *TDevMode*-Record.
dc_MaxExtent	Größtmögliche Papiergröße.
dc_MinExtent	Kleinstmögliche Papiergröße
dc_Papers	Liste unterstützter Papiergrößen.
dc_Papersize	Liste der genauen Abmessungen der unterstützten Papiergrößen.
dc_Size	Das Feld *dmSize* des *TDevMode*-Record.
dc_Version	Spezifikations-Version, die der Gerätetreiber verwendet.

dcb_Xxxx

Diese *Flags* können als Bit-Maske verwendet werden, um die Bitfeldeinstellungen im Feld *Flags* eines *TDCB*-Record abzurufen oder zu setzen.

Konstante	Abgerufenes Bit
dcb_Binary	*fBinary*
dcb_ChEvt	*fChEvt*
dcb_DtrDisable	*fDtrDisable*
dcb_Dtrflow	*fDtrFlow*
dcb_InX	*fInX*
dcb_Null	*fNull*
dcb_OutX	*fOutX*
dcb_OutxCtsFlow	*fOutxCtsFlow*
dcb_OutxDsrFlow	*fOutxDsrFlow*
dcb_Parity	*fParity*
dcb_PeChar	*fPeChar*
dcb_RtsDisable	*fRtsDisable*
dcb_Rtsflow	*fRtsflow*

dde_Xxxx

Diese Flags können als Bit-Maske verwendet werden, um die Bitfeldeinstellungen im Feld *Flags* eines *TDDEAdvise*-, *TDDEData*- oder *TDDEPoke*-Record abzurufen oder zu setzen.

Maske	Bitfeld	Verwendet in folgenden Record-Typen
dde_AckReq	*fAckReq*	*TDDEAdvise, TDDEData*
dde_DeferUpdt	*fDeferUpd*	*TDDEAdvise*
dde_Response	*fResponse*	*TDDEData*
dde_Release	*fRelease*	*TDDEData, TDDEPoke*

Diese Bits werden verwendet, um Bits im Record *TDDEAck* zu setzen.

dde_Ack — Das *fAck*-Bit des Word *TDDEAck.Status*. Sind *lParamLo* und *dde_Ack* 1, wird die Anforderung akzeptiert, sind sie 0, wird die Anforderung zurückgewiesen.

dde_AppReturnCode — Für anwendungsspezifische Rückgabewerte reserviert.

dde_Busy — Das *fBusy*-Bit des Word *TDDEAck.Status*. Wird die Anforderung zurückgewiesen, zeigen die beiden Werte *lParamLo*=1 und *dde_Busy*=1 an, daß die Anwendung nicht auf die Anforderung reagieren kann.

DIB_Xxxx

Diese Konstanten bestimmen den Zugriff auf Farben für die Verwendung in geräteunabhängigen Bitmaps. Sie werden in den Funktionen *CreateDIBitmap*, *CreateDIBPatternBrush*, *GetDIBits*, *SetDIBits*, *SetDIBitsToDevice* und *StretchDIBits* verwendet.

DIB_Pal_Colors — Die Farbtabelle ist ein Array von 16-Bit-Indexen in die aktuell realisierte virtuelle Palette.

DIB_RGB_Colors — Die Farbtabelle enthält explizite RGB-Farbwerte.

dlgc_Xxxx

Diese Codes legen fest, welche Dialogeingaben eine Anwendung anstelle von Windows bearbeitet. Sie werden von *wm_GetDlgCode* zurückgegeben.

dlgc_DefPushButton — Botschaften des Standardaktionsschalters.

dlgc_HasSetSel — *em_SetSel*-Botschaften.

dlgc_Button — Aktionsschalterbotschaften.

dlgc_RadioButton — Schaltfeldbotschaften.

dlgc_UndefPushButton — Aktionsschalterbotschaften.

dlgc_WantAllKeys	Alle Tastatureingabebotschaften.
dlgc_WantArrows	Cursor-Tastenbotschaften.
dlgc_WantChars	*wm_Char*-Botschaften.
dlgc_WantMessage	Alle Tastatureingaben (die Anwendung leitet die Botschaft an das Steuerelement weiter).
dlgc_WantTab	Tab -Tastenbotschaften.

DlgWindowExtra

Soll eine neue Klasse für Dialogfenster aus einer Ressource definiert werden, setzen Sie das Feld *cbWndExtra* des Record *TWndClass* auf *DlgWindowExtra*.

dm_Xxxx

Diese Konstanten werden als Feldauswahlbits verwendet. Jede Konstante repräsentiert ein bestimmtes Bit im Feld *dmFields* eines *TDevMode*-Record. Ist ein bestimmtes Bit gesetzt, zeigt es an, daß das korrespondierende Feld im Record initialisiert wurde.

Konstante	Feld
dm_Color	*dmColor*
dm_Copies	*dmCopies*
dm_DefaultSource	*dmDefaultSource*
dm_Duplex	*dmDuplex*
dm_Orientation	*dmOrientation*
dm_PaperLength	*dmPaperLength*
dm_PaperSize	*dmPaperSize*
dm_PaperWidth	*dmPaperWidth*
dm_PrintQuality	*dmPrintQuality*
dm_Scale	*dmScale*

Diese Konstanten werden im Parameter *Mode* der Funktion *ExtDeviceMode* verwendet, um bestimmte Operationen zu veranlassen. Mode kann einen oder mehrere der folgenden Werte annehmen.

dm_Copy	Schreibt die Gerätetreibereinstellungen in den *TDevMode*-Record, der im Parameter *DevModeOutput* übergeben wird.
dm_Modify	Verändert die Einstellungen des Gerätetreibers. Verwendet die Informationen des *TDevMode*-Records, der im Parameter *DevModeInput* übergeben wird.
dm_Prompt	Erzeugt das Dialogfenster für die Gerätetreibereinstellung und stellt den Drucker nach Maßgabe der Eingaben des Anwenders ein.

dm_Update Aktualisiert die Druckerumgebung sowie die Datei WIN.INI, und zwar gemäß den aktuellen Druckereinstellungen.

dmbin_Xxxx

Diese Konstanten werden im Feld *dmDefaultSource* eines *TDevMode*-Record verwendet. Sie legen fest, welcher Papierschacht angesprochen wird.

dmbin_Auto
dmbin_Cassette
dmbin_Envelope
dmbin_EnvManual
dmbin_LargeCapacity
dmbin_LargeFmt
dmbin_Lower
dmbin_Manual
dmbin_Middle
dmbin_OnlyOne
dmbin_SmallFmt
dmbin_Tractor
dmbin_Upper

dmcolor_Xxxx

Diese Konstanten werden im Feld *dmColor* eines *TDevMode*-Record verwendet. Sie zeigen an, ob der Drucker farbig drucken kann.

dmcolor_Monochrome Drucker kann nur einfarbig drucken

dmcolor_Color Drucker druckt farbig.

dmdup_Xxxx

Diese Konstanten werden im Feld *dmDuplex* eines *TdevMode*-Record verwendet. Sie legen die Art des Duplex-Druckes fest.

dmdup_Horizontal

dmdup_Simplex Nur einseitig drucken.

dmdup_Vertical

dmorient_Xxxx

Diese Konstanten werden im Feld *dmOrientation* eines *TDevMode*-Record verwendet. Sie zeigen die gewünschte Druckorientierung an.

dmorient_Portrait Portrait-Orientierung.

dmorient_Landscape Landscape-Orientierung.

dmpaper_Xxxx

Diese Konstanten werden im Feld *dmPaperSize* eines *TDevMode*-Record verwendet. Sie zeigen die Größe des verwendeten Papiers an.

Konstante	Breite	Höhe	Einheit
dmpaper_10X14	10	14	Zoll
dmpaper_11X17	11	17	Zoll
dmpaper_A3	297	420	mm
dmpaper_A4	210	297	mm
dmpaper_A4Small	210	297	mm
dmpaper_A5	148	210	mm
dmpaper_B4	250	354	mm
dmpaper_B5	182	257	mm
dmpaper_CSheet	C		
dmpaper_DSheet	D		
dmpaper_Env_10	4 1/8	9 1/2	Zoll
dmpaper_Env_11	4 1/2	10 3/8	Zoll
dmpaper_Env_12	4 3/4	11	Zoll
dmpaper_Env_14	5	11 1/2	Zoll
dmpaper_Env_9	3 7/8	8 7/8	Zoll
dmpaper_ESheet	E		
dmpaper_Executive	7 1/2	10	Zoll
dmpaper_Folio	8 1/2	13	Zoll
dmpaper_Ledger	17	11	Zoll
dmpaper_Legal	8 1/2	14	Zoll
dmpaper_Letter	8 1/2	11	Zoll
dmpaper_LetterSmall	8 1/2	11	Zoll
dmpaper_Note	8 1/2	11	Zoll
dmpaper_Quarto	215	275	mm
dmpaper_Statement	5 1/2	8 1/2	Zoll
dmpaper_Tabloid	11	17	Zoll

dmres_Xxxx

Diese Konstanten werden im Feld *dmPrintQuality* eines *TDevMode*-Record verwendet. Sie zeigen die verwendete Auflösung beim Druck an.

dmres_Draft Draft-Modus.

dmres_High Hohe Druckqualität.

dmres_Low Niedrige Druckqualität.

dmres_Medium Mittlere Druckqualität.

drive_Xxxx

Diese Konstanten bestimmen den Typ eines Laufwerks und werden von der Funktion *GetDriveType* zurückgegeben.

drive_Fixed Festplattenlaufwerk.

drive_Remote	Netzlaufwerk.
drive_Removeable	Diskettenlaufwerk.

Drucker-Escape-Codes

Diese Codes werden in der *Escape*-Funktion übergeben, um Anwendungen den direkten Zugriff auch auf diejenigen gerätespezifischen Fähigkeiten zu ermöglichen, die von der grafischen Geräteschnittstelle nicht unterstützt werden.

AbortDoc	Bricht den aktuellen Druckauftrag ab.
BandInfo	Kopiert Informationen über die speziellen Fähigkeiten eines Geräts in einen Record, der an die Funktion Escape übergeben wird. Diese Fähigkeiten beziehen sich auf die Möglichkeit, die Ausgabe in einzelne Abschnitte aufzuteilen, die vor der Ausgabe in einem Metafile gespeichert wird.
Begin_Path	Öffnet einen Pfad, der ein Polygon oder einen Linienzug aus verbundenen Punkten darstellt. Pfade werden verwendet, um die Befehlsübergabe von Befehlen an PostScript-Drucker zu erleichtern.
Clip_To_Path	Definiert eine Clipping-Region auf Grundlage des aktuellen Pfads.
DeviceData	Wie *PassThrough*.
DraftMode	Schaltet den Draft-Modus (EDV-Qualität) an oder aus.
DrawPatternRect	Zeichnet ein gemustertes, ein schwarzes oder ein Graustufenrechteck auf einem PCL-kompatiblen Hewlett-Packard Drucker.
EnableDuplex	Schaltet die Druckerfunktion zum beidseitigen Druck ein.
EnablePairKerning	Aktiviert oder deaktiviert die Kerning-Funktion des Druckers.
EnableRelativeWidths	Aktiviert oder deaktiviert die relative Zeichenbreite. Ist diese Option deaktiviert, ist die Breite eines Strings identisch mit der Summe der Zeichenbreiten. Ist die Option aktiviert, müssen Sie auf die Breitentabelle der Schrift zugreifen und die Breite des Strings berechnen.
End_Path	Beendet einen Pfad (vgl. *Begin_Path*).
EndDoc	Beendet einen Druckauftrag, der durch *StartDoc* gestartet wurde (vgl. *StartDoc*).
EnumPaperBins	Ruft Informationen über die Papierschächte des Drukkers ab. Sie sollten besser die Konstante *GetSetPaper-*

	Bins verwenden. Dieser *Escape*-Code wurde nur aus Gründen der Abwärtskompatibilität übernommen.
EnumPaperMetrics	Ruft die Anzahl unterstützter Papiertypen oder die Größe der druckbaren Rechtecke ab. Sie sollten besser die Funktion *ExtDeviceMode* verwenden. Dieser *Escape*-Code wurde nur aus Gründen der Abwärtskompatibilität übernommen.
EPSPrinting	Unterdrückt den Steuerungsabschnitt des PostScript-Headers und unterbindet dadurch sämtliche GDI-Aufrufe.
Ext_Device_Caps	Ruft Informationen über gerätespezifische Druckerfähigkeiten ab, die über die Rückgabe der Funktion *GetDeviceCaps* hinausgehen.
FlushOutput	Entleert den Gerätepuffer.
GDIExtTextOut	Entspricht der Funktion *ExtTextOut*.
GDIStretchBlt	Entspricht der Funktion *StretchBlt*.
GetColorTable	Ruft einen RGB-Farbtabelleneintrag ab.
GetExtendedTextMetrics	Ruft die erweiterten Textabmessungen der ausgewählten Schrift ab.
GetExtentTable	Ruft die Ausdehnung (Breite) eines Zeichenbereichs im aktuellen Zeichensatz der Schrift ab.
GetPairKernTable	Ruft die Zeichenpaar-Kerning-Tabelle ab.
GetPhysPageSize	Ruft die physikalische Seitengröße ab.
GetPrintingOffset	Ruft den Offset der oberen linken Position auf einer Seite ab, an der das Drucken oder Zeichnen begonnen werden soll.
GetScalingFactor	Ruft den Skalierungsfaktor des Druckers in x- und y-Richtung ab.
GetSetPaperBins	Ruft die Anzahl der Papierschächte des Druckers ab und setzt den aktuellen Schacht ein.
GetSetPaperMetrics	Ruft die Papierabmessungsinformationen des Druckers ab und stellt sie neu ein. Sie sollten besser die Funktion *ExtDeviceMode* verwenden. Dieser *Escape*-Code wurde nur aus Gründen der Abwärtskompatibilität übernommen.
GetSetPrintOrient	Ruft die aktuelle Papierausrichtung ab oder stellt sie neu ein. Sie sollten besser die Funktion *ExtDeviceMode* verwenden. Dieser *Escape*-Code wurde nur aus Gründen der Abwärtskompatibilität übernommen.

GetTechnology	Ruft einen String ab, der die grundlegende Arbeitsweise des Druckers bezeichnet, z.B. PostScript.
GetTrackKernTable	Ruft die Track-Kerning-Tabelle der aktuell ausgewählten Schrift ab.
GetVectorBrushSize	Ruft die Breite des Plotterstifts in Geräteeinheiten ab, der zum Ausfüllen geschlossener Formen verwendet wird.
GetVectorPenSize	Ruft die Breite des Plotterstifts in Geräteeinheiten ab, der für Schraffurmuster verwendet wird.
MFComment	Fügt einen Kommentar zu einem Metafile hinzu.
NewFrame	Weist den Drucker an, auf einer neuen Seite fortzufahren.
NextBand	Teilt dem Gerätetreiber mit, daß die Ausgabe eines Abschnitts in ein Metafile abgeschlossen wurde.
PassThrough	Erlaubt einer Anwendung die unmittelbare Übergabe von Daten an den Drucker.
QueryEscSupport	Ermittelt, ob der Gerätetreiber einen bestimmten *Escape*-Code unterstützt.
Restore_CTM	Stellt die vorher gesicherte, aktuelle Transformationsmatrix wieder her.
Save_Ctm	Sichert die aktuelle Transformationsmatrix.
SelectPaperSource	Wird von *GetSetPaperBins* ersetzt und wurde nur aus Gründen der Abwärtskompatibilität übernommen.
Set_Arc_Direction	Bestimmt die Richtung, in der Kreisbögen durch die Funktion *Arc* gezeichnet werden.
Set_Background_Color	Ruft die Hintergrundfarbe eines Geräts ab oder stellt sie neu ein.
Set_Bounds	Stellt das eingrenzende Rechteck für das darzustellende Bild ein.
Set_Clip_Box	Setzt ein Clipping-Rechteck oder stellt das vorhergehende wieder her.
Set_Poly_Mode	Stellt den Poly-Modus ein, der festlegt, wie Aufrufe der Funktionen *Polygon* und *PolyLine* interpretiert werden.
Set_Screen_Angle	Setzt den aktuellen Neigungswinkel der Rasterung und simuliert so die Neigung eines photographischen Farbfilters um die Farbtrennung für eine Grundfarbe zu ermöglichen.

Set_Spread	Setzt den Umfang, um den nichtweiße Grundelemente erweitert werden, um Fehler bei der Reproduktion auszugleichen.
SetAbortProc	Setzt die Abbruchfunktion für den aktuellen Druckauftrag.
SetAllJustValues	Setzt die Ausrichtungswerte für die Textausgabe.
SetColorTable	Setzt einen RGB-Farbtabelleneintrag.
SetCopyCount	Bestimmt die Anzahl der Kopien, die von jeder Seite gedruckt werden sollen.
SetKernTrack	Bestimmt, welcher Kerning-Track verwendet werden soll.
SetLineJoin	Legt fest, wie schneidende Linien miteinander verbunden werden: mit abgerundeten, eckigen oder bis zur Spitze verlängerten Ecken.
SetMiterLimit	Setzt den Gehrungswinkel, ab dem Linienverbindungen nicht bis zur Spitze verlängert, sondern eckig abgeschnitten werden.
StartDoc	Teilt dem Gerätetreiber mit, daß ein neuer Druckauftrag beginnt.
Transform_CTM	Verändert die aktuelle Transformationsmatrix.

ds_Xxxx

Diese Konstanten werden verwendet, um den Stil eines Dialogfensters festzulegen, wenn es mit *CreateWindow* oder *CreateWindowEx* erzeugt wird.

ds_AbsAlign	Bestimmt, daß das Dialogfenster relativ zum Bildschirmursprung und nicht zum Besitzer des Dialogfensters ausgerichtet wird.
ds_LocalEdit	Bestimmt, daß Textspeicher für Editierfelder im lokalen Datensegment einer Anwendung reserviert wird. Dies erlaubt die Nutzung der Botschaften *em_GetHandle* und *em_SetHandle*. Wenn dieser Stil nicht vorgegeben ist, befindet sich der Datenbereich für Editierfelder in einem separaten globalen Datenblock.
ds_ModalFrame	Bestimmt ein Dialogfenster mit modalem Rahmen. Dieser Stil kann mit *ws_Caption* und *ws_SysMenu* kombiniert werden, um ein Dialogfenster mit einer Überschrift und einem Systemmenü zu erzeugen.
ds_NoIdleMsg	Bestimmt, daß Windows keine *wm_EnterIdle*-Botschaft an den Besitzer übergibt, während das Dialogfenster angezeigt wird.

ds_SetFont Bestimmt, daß eine andere Schrift als die Systemschrift verwendet wird. Ist dieses Flag gesetzt, muß die *FontInfo*-Datenstruktur (beschrieben in den folgenden Absätzen) folgen. Wenn Windows ein Dialogfenster mit diesem Attribut erzeugt, sendet Windows die Botschaft *wm_SetFont* vor der Erstellung der Steuerelemente an das Dialogfenster.

ds_SysModal Bestimmt ein systemmodales Fenster. Alle Windows-Anwendungen werden angehalten, während dieses Dialogfenster angezeigt wird.

dt_Xxxx

Folgende Konstanten repräsentieren den Typ des verwendeten Geräts.

dt_CharStream Zeichen-Stream.

dt_DispFile Bildschirmdatei.

dt_MetaFile Metafile.

dt_Plotter Vektorplotter.

dt_RasCamera Rasterkamera.

dt_RasDisplay Rasterbildschirm.

dt_RasPrinter Rasterdrucker.

Folgende Konstanten legen die Formatierungsoptionen für die Funktion *DrawText* fest.

dt_Bottom Bestimmt an der Grundlinie ausgerichteten Text. Dieser Wert muß mit *dt_SingleLine* kombiniert werden.

dt_CalcRect Ermittelt Höhe und Breite des Rechtecks. Gibt es mehrere Textzeilen, verwendet *DrawText* die Breite des durch *Rect* bezeichneten Rechtecks und erweitert dessen Basis, um auch die letzte Textzeile einzuschließen. Existiert nur eine Textzeile, wird die rechte Seite des Rechtecks so angepaßt, daß auch das letzte Zeichen der Textzeile eingeschlossen wird. In jedem Fall gibt *DrawText* die Höhe des Textes zurück, stellt ihn aber nicht dar.

dt_Center Zentriert den Text horizontal.

dt_ExpandTabs Erweitert die Tabulatoren. Der Vorgabewert von Zeichen je Tabulator ist acht.

dt_ExternalLeading Schließt die Oberlinien der Schrift ein. Normalerweise werden diese Oberlinien in die Höhe einer Textzeile nicht eingeschlossen.

dt_Left Stellt Text linksbündig dar.

dt_NoClip Zeichnet ohne Clipping. DrawText arbeitet schneller, wenn *dt_NoClip* verwendet wird.

dt_NoPrefix Schaltet die Bearbeitung von Präfix-Zeichen aus. Normalerweise interpretiert *DrawText* das mnemonische Präfix-Zeichen »&« als Anweisung zum Unterstreichen des nachfolgenden Zeichens. Das mnemonische Präfix-Zeichen »&&« wird dann als Anweisung zum Druck eines einzelnen »&« interpretiert.
Das Setzen von *dt_NoPrefix* schaltet diese Bearbeitung aus.

dt_Right Richtet Text rechtsbündig aus.

dt_SingleLine Legt eine einzige Textzeile fest. Wagenrücklauf und Zeilenvorschub führen nicht zum Zeilenumbruch.

dt_TabStop Setzt Tabulatoren. Das höherwertige Word des Parameters *Format* bestimmt die Anzahl der Zeichen pro Tabulator. Der Standardvorgabewert ist acht.

dt_Top Bestimmt bei einzelnen Textzeilen die Ausrichtung des Textes an der Oberlinie.

dt_VCenter Legt bei einzelnen Textzeilen vertikale Zentrierung fest.

dt_WordBreak Legt den Zeilenumbruch fest: Zeilen werden wortweise umbrochen, wenn ein Wort nicht mehr in die vorhergehende Zeile paßt. Dabei wird die durch den Parameter *Rect* festgelegte Kante des Rechtecks zugrunde gelegt. Auch eine Wagenrücklauf-Zeilenvorschub-Sequenz führt zu einem Zeilenumbruch.

en_Xxxx

Die folgenden Benachrichtigungscodes sind Editierfeldern zugeordnet:

en_Change Zeigt an, daß der Anwender eine Aktion durchgeführt hat, die möglicherweise den Inhalt des Textes verändert hat.

en_ErrSpace Zeigt an, daß kein zusätzlicher Platz im Editierfeld vorhanden ist.

en_HScroll Zeigt an, daß der Benutzer die horizontale Bildlaufleiste eines Editierfeldes mit der Maus angeklickt hat. Das übergeordnete Fenster wird vor dem Aktualisieren des Bildschirms benachrichtigt.

en_KillFocus Zeigt an, daß das Editierfeld den Fokus verloren hat.

en_MaxText Zeigt an, daß die aktuelle Einfügung die festgelegte Anzahl von Zeichen für dieses Editierfeld überschritten hat.

en_SetFocus Zeigt an, daß das Editierfeld den Fokus erhalten hat.

en_Update Zeigt an, daß das Editierfeld veränderten Text darstellt.

en_VScroll Zeigt an, daß der Anwender die vertikale Bildlaufleiste des Editierfeldes mit der Maus angeklickt hat. Das übergeordnete Fenster wird vor dem Aktualisieren des Bildschirms benachrichtigt.

es_Xxxx

Diese Konstanten werden verwendet, um den Stil von Editierfeldern festzulegen, wenn diese mit *CreateWindow* oder *CreateWindowEx* erzeugt werden.

es_AutoHScroll Rollt automatisch Text um 10 Zeichen nach rechts, wenn der Benutzer am Ende der Zeile ein Zeichen eintippt. Wenn der Benutzer die Return -Taste drückt, rollt das Steuerelement den ganzen Text zur Nullposition zurück.

es_AutoVScroll Rollt automatisch Text um eine Seite nach oben, wenn der Benutzer in der letzten Zeile Return drückt.

es_Center Zentriert Text in einem mehrzeiligen Editierfeld. Kann nur mit *es_MultiLine* verwendet werden.

es_Left Richtet Text linksbündig aus. Kann nur mit *es_MultiLine* verwendet werden.

es_LowerCase Wandelt alle Zeichen in Kleinbuchstaben um, sowie sie im Editierfeld eingegeben werden.

es_MultiLine Bestimmt eine mehrzeiliges Editierfeld (die Voreinstellung ist einzeilig). Wenn der Stil *es_AutoVScroll* angegeben wird, zeigt das Editierfeld so viele Zeilen wie möglich und rollt vertikal, wenn der Benutzer die [Return]-Taste drückt. Wenn *es_AutoVScroll* nicht gesetzt ist, zeigt das Editierfeld so viele Zeilen wie möglich und gibt einen Warnton aus, falls Return gedrückt wird, da keine weiteren Zeilen angezeigt werden können.
Wenn der Stil *es_AutoHScroll* gesetzt wird, rollt das mehrzeilige Editierfeld automatisch horizontal, wenn das Caret hinter den rechten Rand des Steuerelements gelangt. Um eine neue Zeile zu beginnen, muß der Benutzer Return drücken. Wenn *es_AutoHScroll* nicht gesetzt ist, bricht das Steuerelement, falls nötig, auto-

matisch Wörter auf den Anfang der nächsten Zeile um; eine neue Zeile wird auch dann begonnen, wenn Return gedrückt wird.
Die Position des Zeilenumbruchs wird von der Größe des Fensters bestimmt. Wenn die Größe des Fensters sich ändert, verändert sich die Position des Zeilenumbruchs und der Text wird neu dargestellt.
Mehrzeilige Editierfelder können Bildlaufleisten haben. Ein Editierfeld mit Bildlaufleisten bearbeitet seine eigenen Bildlaufleistenbotschaften. Editierfelder ohne Bildlaufleisten rollen wie oben beschrieben und bearbeiten jede Bildlaufbotschaft, die vom übergeordneten Fenster gesendet wird.

es_NoHideSel — Normalerweise macht ein Editierfeld die Auswahl unsichtbar, wenn das Steuerelement den Fokus verliert und invertiert das Auswahlfenster, wenn das Steuerelement den Fokus erhält.
Die Angabe von *es_NoHideSel* löscht diese voreingestellte Aktion.

es_OEMConvert — Text, der im Editierfeld eingegeben wird, wird vom ANSI-Zeichensatz in den OEM-Zeichensatz und dann zurück in ANSI umgewandelt. Dies sorgt für eine genaue Zeichenumwandlung, wenn die Anwendung die Funktion *AnsiToOem* aufruft, um einen ANSI-String im Editierfeld in OEM-Zeichen umzuwandeln. Dieser Stil ist für Editierfelder hilfreich, die Dateinamen enthalten.

es_Password — Stellt alle Zeichen als Stern (*) dar, sobald sie im Editierfeld eingegeben werden. Eine Anwendung kann die Botschaft *em_SetPassWordChar* verwenden, um das Zeichen zu verändern, das angezeigt wird.

es_Right — Richtet Text in einem mehrzeiligen Editierfeld rechtsbündig aus.

es_UpperCase — Wandelt alle Zeichen in Großbuchstaben um, sobald sie in das Editierfeld eingegeben werden.

Escape-Konstanten für serielle Schnittstellen

Diese Konstanten bestimmen den Code der Funktion, die von einer Schnittstelle ausgeführt werden soll. Sie werden in Aufrufen der Funktion *EscapeCommFunction* verwendet.

ClrDTR — Löscht das Data-Terminal-Ready-Signal (DTR).

ClrRTS — Löscht das Request-To-Sent-Signal (RTS).

ResetDev	Wenn möglich, wird die Schnittstelle in den Ausgangszustand zurückgesetzt.
SetDTR	Sendet das Data-Terminal-Ready-Signal.
SetRTS	Sendet das Request-To-Send-Signal.
SetXOFF	Beeinflußt die Übertragung so, als wäre das Zeichen XOFF empfangen worden.
SetXON	Beeinflußt die Übertragung so, als wäre das Zeichen XON empfangen worden.

eto_Xxxx

Diese Optionen bestimmen die Hintergrunddarstellung durch die Funktion *ExtTextOut*.

eto_Clipped	Der Text wird dem umgebenden Rechteck angepaßt.
eto_Opaque	Das umgebende Rechteck wird mit der Hintergrundfarbe ausgefüllt.

ev_Xxxx

Diese Konstanten dienen als Zeiger in die Ereignismaske einer Schnittstelle. Außerdem werden sie als Rückgabewert der Funktion *SetCommEventMask* verwendet. Die folgende Aufstellung beschreibt, unter welchen Umständen die bezeichneten Maskenbits gesetzt sind.

ev_Break	Gesetzt, wenn eine Unterbrechung bei der Eingabe erkannt wurde.
ev_CTS	Gesetzt, wenn das Signal Clear-to-send (CTS) den Status wechselt.
ev_DSR	Gesetzt, wenn das Signal Data-set-ready (DSR) den Status wechselt.
ev_Err	Gesetzt, wenn ein Leitungsstatusfehler auftritt. Leitungsstatusfehler sind *ce_Frame*, *ce_Overrun* und *ce_RXParity*.
ev_PErr	Gesetzt, wenn ein Druckerfehler an einer parallelen Schnittstelle erkannt wurde. Fehler sind *ce_DNS*, *ce_IOE*, *ce_OOP* und *ce_PTO*.
ev_Ring	Gesetzt, wenn ein Anrufsignal erkannt wurde.
ev_RLSD	Gesetzt, wenn das Signal Receive-line-signal-detect (RLSD) den Status ändert.
ev_RXChar	Gesetzt, wenn ein Zeichen empfangen und in die Warteschlange gestellt wurde.

ev_RXFlag Gesetzt, wenn das Ereigniszeichen empfangen und in die Empfangs-Warteschlange gestellt wurde. Das Ereigniszeichen wird im Gerätesteuerblock des Gerätes bestimmt.

ev_TXEmpty Gesetzt, wenn das letzte Zeichen in der Sende-Warteschlange gesendet wurde.

ff_Xxxx

Diese Konstanten legen den gewünschten Zeichensatz für Schriften fest, die mit der Funktion *CreateFont* erzeugt werden.

ff_Decorative Ungewöhnliche Schriften, z.B. Old English.

ff_DontCare Nicht wichtig oder unbekannt.

ff_Modern Schriften mit konstantem Zeichenabstand, mit oder ohne Serifen. Schriften mit konstantem Zeichenabstand sind normalerweise modern, z.B. Pica, Elite und Courier.

ff_Roman Schriften mit variablem Zeichenabstand (Proportionalschriften) und mit Serifen, z.B. Times Roman, Palatino, Century Schoolbook.

ff_Script Schriften, die wie Handschrift aussehen, z.B. Script und Cursive.

ff_Swiss Serifenlose Schriften mit variablem Zeichenabstand (Proportionalschriften), z. B. Helvetica.

FloodFill-Flags

Diese Flags legen in der Funktion *ExtFloodFill* den Füllmodus fest.

FloodFillBorder Der auszumalende Bereich wird von der durch *Color* festgelegten Farbe begrenzt. Diese Art des Ausmalens entspricht der Funktion *FloodFill*.

FloodFillSurface Der auszumalende Bereich wird durch die in *Color* festgelegte Farbe definiert. Das Ausmalen wird nach außen in allen Richtungen solange fortgesetzt, wie die festgelegte Farbe angetroffen wird.
Diese Art des Ausmalens ist besonders bei mehrfarbigen Rändern hilfreich.

fw_Xxxx

Diese Konstanten bestimmen die gewünschte Schriftstärke für Schriften, die mit der Funktion *CreateFont* erzeugt werden.

Die Flags sind in aufsteigender Reihenfolge, von der dünnsten zur stärksten Schrift, aufgelistet.

fw_DontCare
fw_Thin
fw_ExtraLight, fw_UltraLight
fw_Light
fw_Normal, fw_Regular
fw_Medium
fw_SemiBold, fw_DemiBold
fw_Bold,
fw_ExtraBold, fw_UltraBold
fw_Heavy, fw_Black

gcl_Xxxx

Diese Konstanten bestimmen den Byte-Offset der Fensterklasseninformationen, die mit der Funktion *GetClassLong* oder *SetClassLong* abgerufen bzw. gesetzt werden.

gcl_MenuName	Zeiger auf den Menünamen (nur *SetClassLong*).
gcl_WndProc	Zeiger auf die Fensterfunktion.

gcw_Xxxx

Diese Konstanten bestimmen den Byte-Offset der Fensterklasseninformationen, die mit der Funktion *GetClassWord* oder *SetClassWord* abgerufen bzw. gesetzt werden.

gcw_CBClsExtra	Setzt zwei neue Bytes zusätzlicher Fensterklassendaten.
gcw_CBWndExtra	Setzt zwei neue Bytes zusätzlicher Fensterdaten.
gcw_HBRBackground	Setzt ein neues Handle auf einen Hintergrundpinsel.
gcw_HCursor	Setzt ein neues Handle auf einen Mauszeiger.
gcw_HIcon	Setzt ein neues Handle auf ein Symbol.
gcw_HModule	Handle auf das Modul (nur *GetClassWord*).
gcw_Style	Setzt ein neues Stilbit für die Fensterklasse.

Gerätefähigkeits-Konstanten

Diese Konstanten entsprechen bestimmten Fähigkeiten eines Gerätekontextes. Sie werden in der Funktion *GetDeviceCaps* verwendet.

AspectX	Relative Breite des zum Zeichnen einer Linie verwendeten Pixels eines Geräts.

AspectXY	Diagonale Abmessung des zum Zeichnen einer Linie verwendeten Pixels eines Geräts.
AspectY	Relative Höhe des zum Zeichnen einer Linie verwendeten Pixels eines Geräts.
BitsPixel	Anzahl benachbarter Farb-Bits eines Pixels.
ClipCaps	Flag, das die Clipping-Fähigkeit des Geräts anzeigt. Die möglichen Werte sind in diesem Kapitel im Abschnitt »cp_Xxxx« beschrieben.
ColorRes	Gegenwärtige Farbauflösung des Geräts in Bits je Pixel. Dieser Index ist nur dann gültig, wenn der Gerätetreiber das *rc_Palette*-Bit im *RasterCaps*-Index setzt. Er ist nur verfügbar, wenn die Versionsnummer des Treibers 3.0 oder höher ist.
CurveCaps	Eine Bitmaske, welche die Bogenzeichnungsfähigkeit des Geräts anzeigt. Die möglichen Werte sind in diesem Kapitel im Abschnitt »cc_Xxxx« beschrieben.
DriverVersion	Versionsnummer, zum Beispiel $100 für 1.0.
HorzRes	Breite der Anzeige in Pixel.
HorzSize	Breite der physikalischen Anzeige in Millimetern.
LineCaps	Eine Bitmaske, welche die Linienzeichnungsfähigkeiten des Geräts anzeigt. Die möglichen Werte sind in diesem Kapitel im Abschnitt »lc_Xxxx« beschrieben.
LogPixelsX	Anzahl der Pixel je horizontalem Zoll.
LogPixelsY	Anzahl der Pixel je vertikalem Zoll.
NumBrushes	Anzahl der gerätespezifischen Pinsel.
NumColors	Anzahl der Einträge in der Farbtabelle des Geräts.
NumFonts	Anzahl der gerätespezifischen Schriften.
NumMarkers	Anzahl der Markierungssymbole.
NumPens	Anzahl der gerätespezifischen Stifte.
NumReserved	Anzahl der reservierten Einträge in der Systempalette. Dieser Index ist nur dann gültig, wenn der Gerätetreiber das *rc_Palette*-Bit im *RasterCaps*-Index setzt. Er ist nur verfügbar, wenn die Versionsnummer des Treibers 3.0 oder höher ist.
PDeviceSize	Benötigte Größe des Geräte-Deskriptors.
Planes	Anzahl der Farbebenen.
PolygonalCaps	Eine Bitmaske, welche die Fähigkeit des Geräts zum Zeichnen von Vielecken anzeigt. Die möglichen Wer-

te sind in diesem Kapitel in der Übersicht »pc_Xxxx« beschrieben.

RasterCaps — Wert, der die Scan-Fähigkeiten des Geräts anzeigt. Die möglichen Werte sind in diesem Kapitel in der Übersicht »rc_Xxxx« beschrieben.

SizePalette — Anzahl der Einträge in der Systempalette. Dieser Index ist nur gültig, wenn der Gerätetreiber das *rc_Palette*-Bit im *RasterCaps*-Index setzt. Er ist nur verfügbar, wenn die Versionsnummer des Treibers 3.0 oder höher ist.

Technology — Gerätetechnik. Die möglichen Werte sind in diesem Kapitel in der Übersicht »dt_Xxxx« beschrieben.

TextCaps — Eine Bitmaske, welche die Fähigkeit zur Textdarstellung des Geräts anzeigt. Die möglichen Werte sind in diesem Kapitel in der Übersicht »tc_Xxxx« beschrieben.

VertRes — Höhe der Anzeige in Scan-Zeilen.

VertSize — Höhe der physikalischen Anzeige in Millimetern.

gmem_Xxxx

Diese Flags bestimmen Eigenschaften des globalen Speicherblocks, der mit der Funktion *GlobalAlloc* oder *GlobalReAlloc* erzeugt wird. Sie werden außerdem mit den Funktionen *GlobalFlags* und *GetFreeSpace* verwendet.

gmem_DDEShare — Reserviert gemeinsam benutzbaren Speicherplatz. Wird nur für dynamischen Datenaustausch (DDE) verwendet. Es ist jedoch zu beachten, daß Windows automatisch Speicher verwirft, der mit diesem Attribut reserviert wurde, wenn die Anwendung, die diesen Speicher belegte, die Ausführung beendet.

gmem_Discardable — Reserviert verwerfbaren Speicher. Kann nur mit *gmem_Moveable* benutzt werden.

gmem_Discarded — Der Block wurde verworfen (nur *GlobalFlags*).

gmem_Fixed — Belegt festen Speicher.

gmem_LockCount — Wird dieser Wert mit dem niederwertigen Byte des Rückgabewerts der Funktion *GlobalFlags* kombiniert, wird der Referenzzähler des Speicherblocks zurückgegeben (nur *GlobalFlags*).

gmem_Lower — Wie *gmem_Not_Banked.*

gmem_Modify — Die Flags des globalen Speichers werden verändert, wenn dieses Flag eingeschlossen wird (nur *GlobalReAlloc*).

gmem_Moveable	Reserviert verschiebbaren Speicher. Kann nicht mit *gmem_Fixed* benutzt werden.
gmem_NoCompact	Verdichtet und verwirft nicht, um den Belegungsbedarf zu befriedigen.
gmem_NoDiscard	Verwirft nicht, um den Belegungsbedarf zu befriedigen.
gmem_Not_Banked	Belegt Speicher, der nicht in Bänke aufgeteilt ist. Kann nicht mit *gmem_Notify* benutzt werden.
gmem_Notify	Ruft die Benachrichtigungsroutine auf, falls das Speicherobjekt jemals verworfen wird.
gmem_Share	Wie *gmem_DDEShare*.
gmem_ZeroInit	Initialisiert Speicherinhalte mit 0.

gw_Xxxx

Diese Konstanten bestimmen die Beziehung zwischen dem gewünschten Fenster und dem Fenster, das in Aufrufen der Funktionen *GetNextWindow* und *GetWindow* vorgegeben wird. Die Funktion *GetNextWindow* akzeptiert nur die Werte *gw_HWndNext* und *gw_HWndPrev*.

gw_Child	Bezeichnet das erste untergeordnete Fenster des Fensters.
gw_HWndFirst	Gibt das erste Nachbarfenster eines untergeordneten Fensters an. Existiert kein solches Fenster, wird das erste Hauptfenster in der Liste angezeigt.
gw_HWndLast	Gibt das letzte Nachbarfenster für ein untergeordnetes Fenster oder das letzte Hauptfenster in der Liste an.
gw_HWndNext	Gibt das Fenster an, das dem vorgegebenen Fenster in der Liste des Fenster-Managers folgt.
gw_HWndPrev	Gibt das vorherige Fenster in der Liste des Fenster-Managers an.
gw_Owner	Bezeichnet den Besitzer des Fensters.

gwl_Xxxx

Diese Konstanten bestimmen den Byte-Offset des Fensterattributs, das durch *GetWindowLong* oder *SetWindowLong* abgerufen oder verändert werden soll.

gwl_EXStyle	Erweiterter Fensterstil.
gwl_Style	Fensterstil.
gwl_WndProc	Zeiger auf die Fensterfunktion.

gww_Xxxx

Diese Konstanten bestimmen den Byte-Offset der Fensterinformation, die durch die Funktionen *GetWindowWord* oder *SetWindowWord* abgerufen oder verändert werden soll.

gww_HInstance — Instanz-Handle des Moduls, das das Fenster besitzt.

gww_HWndParent — Handle des übergeordneten Fensters, falls vorhanden. Die Funktion *SetParent* ändert das übergeordnete Fenster eines untergeordneten Fensters. Eine Anwendung sollte nicht die Funktion *SetWindowLong* aufrufen, um das übergeordnete Fenster eines untergeordneten Fensters zu ändern (nur *GetWindowWord*).

gww_ID — Steuerelement-ID des untergeordneten Fensters.

help_Xxxx

Diese Konstanten zeigen der Windows-Hilfefunktion an, welche Art von Hilfe von der Anwendung angefordert wird. Sie werden in Aufrufen der Funktion *WinHelp* verwendet.

Help_Context — Zeigt Hilfe für einen bestimmten Kontext an, der durch einen vorzeichenlosen 32-Bit-Integer in *Data* bezeichnet wird.

Help_HelpOnHelp — Zeigt Hilfe für die Benutzung der Hilfeanwendung selbst an. Wenn der Parameter *Command* auf *Help_HelpOnHelp* gesetzt wird, ignoriert *WinHelp* die Parameter *HelpFile* und *Data*.

Help_Index — Zeigt den Index der angegebenen Hilfedatei an. Eine Anwendung sollte diesen Wert nur für Hilfedateien mit einem einzigen Index benutzen. Sie sollte diesen Wert nicht mit *Help_SetIndex* benutzen.

Help_Key — Zeigt Hilfe für ein einzelnes Schlüsselwort an, das durch einen String bezeichnet wird, auf den *Data* zeigt.

Help_MultiKey — Zeigt Hilfe für ein Schlüsselwort in einer veränderbaren Schlüsselworttabelle an.

Help_Quit — Teilt der Hilfeanwendung mit, daß die angegebene Hilfedatei nicht mehr in Gebrauch ist.

Help_SetIndex — Setzt den durch den Parameter *Data* bestimmten Kontext als aktuellen Index für die durch den Parameter *HelpFile* angegebene Hilfedatei. Dieser Index bleibt aktuell, bis der Benutzer auf eine andere Hilfedatei zugreift. Um sicherzustellen, daß der richtige Index

gesetzt bleibt, sollte die Anwendung nach jedem *WinHelp*-Aufruf mit auf *Help_Context* gesetzten *Command* *WinHelp* aufrufen, wobei dann *Command* auf *Help_SetIndex* gesetzt sein sollte (und *Data* den zugehörigen Kontextbezeichner angibt). Eine Anwendung sollte diesen Wert nur für Hilfedateien mit mehr als einem Index benutzen. Sie sollte diesen Wert nicht mit *Help_Index* anwenden.

Hintergrund-Modi

Diese Konstanten bestimmen die Darstellung des Hintergrunds, wenn Text oder Grafik mit einem Schraffurpinsel dargestellt wird. Sie werden mit den Funktionen *GetBkMode* und *SetBkMode* verwendet.

Opaque	Hintergrund wird mit der aktuellen Hintergrundfarbe ausgefüllt, bevor der Text oder die Schraffur gezeichnet wird.
Transparent	Hintergrund bleibt unberührt.

hs_Xxxx

Die folgenden Konstanten bestimmen den Schraffurstil für Pinsel. Sie werden mit der Funktion *CreateHatchBrush* verwendet.

hs_BDiagonal	45-Grad abwärts schraffiert (von links nach rechts: \ \ \ \ ).
hs_Cross	Horizontal und vertikal überkreuz schraffiert (++++).
hs_DiagCross	45-Grad überkreuz schraffiert (xxxx).
hs_FDiagonal	45-Grad aufwärts schraffiert (von links nach rechts: / / / /).
hs_Horizontal	Horizontal schraffiert (- - - -).
hs_vertical	Vertikal schraffiert (| | | |).

htXxxx

Diese Codes zeigen die Position des Mauszeigers im Verhältnis zu einem Fenster an und werden in vielen Botschaften übergeben, die den Mauszeiger positionieren oder bewegen. In Betracht kommen unter anderen die Botschaften *wm_MouseActivate, wm_SetCursor* und *wm_NCHitTest*.

htBottom	Auf der unteren horizontalen Grenzlinie des Fensters.
htBottomLeft	Auf der unteren linken Ecke der Fenstergrenze.
htBottomRight	Auf der unteren rechten Ecke der Fenstergrenze.

htCaption	In einem Titelbereich.
htClient	In einem Client-Bereich.
htError	Wie *htNoWhere*. Zusätzlich erzeugt die Funktion *DefWindowProc* einen Warnlaut, um einen Fehler anzuzeigen.
htGrowBox	In einem Größenveränderungsfeld.
htHScroll	In der horizontalen Bildlaufleiste.
htLeft	Auf der linken Grenzlinie des Fensters.
htMenu	In einem Menübereich.
htNoWhere	Auf dem Bildschirmhintergrund oder einer Trennlinie zwischen zwei Fenstern.
htReduce	In einem Verkleinerungsfeld.
htRight	Auf der rechten Grenzlinie des Fensters.
htSize	Wie *htGrowBox*.
htSysMenu	In einem Menüfeld (Schließfeld in untergeordneten Fenstern).
htTop	Auf der oberen horizontalen Grenzlinie des Fensters.
htTopLeft	Auf der oberen linken Ecke der Fenstergrenze.
htTopRight	Auf der oberen rechten Ecke der Fenstergrenze.
htTransparent	In einem Fenster, auf dem ein anderes Fenster liegt.
htVScroll	In der vertikalen Bildlaufleiste.
htZoom	In einem Vergrößerungsfeld.

id_Xxxx

Die folgenden Konstanten werden als Rückgabewerte der Funktion *MessageBox* verwendet. Sie zeigen das Ergebnis des Meldungsfensters an.

id_Abort	Aktionsschalter »Beenden« gedrückt.
id_Cancel	Aktionsschalter »Abbrechen« gedrückt.
id_Ignore	Aktionsschalter »Ignorieren« gedrückt.
id_No	Kein Aktionsschalter gedrückt.
id_Ok	Aktionsschalter »OK« gedrückt.
id_Retry	Aktionsschalter »Wiederholen« gedrückt.
id_Yes	Aktionsschalter »Ja« gedrückt.

idc_Xxxx

Diese Bezeichner beziehen sich auf die vordefinierten Windows-Mauszeiger. Die Mauszeiger werden mit der Funktion *LoadCursor* geladen.

idc_Arrow	Standardpfeilzeiger.
idc_Cross	Fadenkreuzzeiger.
idc_IBeam	I-Balkenzeiger in Texten.
idc_Icon	Leeres Symbol.
idc_Size	Vierpfeilzeiger. Die Pfeile zeigen nach Norden, Osten, Süden und Westen.
idc_SizeNESW	Doppelpfeilzeiger. Die Pfeile zeigen nach Nordost und Südwest.
idc_SizeNS	Doppelpfeilzeiger. Die Pfeile zeigen nach Norden und Süden.
idc_SizeNWSE	Doppelpfeilzeiger. Die Pfeile zeigen nach Nordwest und Südost.
idc_SizeWE	Doppelpfeilzeiger. Die Pfeile zeigen nach Westen und Osten.
idc_UPArrow	Vertikaler Pfeilzeiger.
idc_Wait	Sanduhrzeiger.

idi_Xxxx

Diese Bezeichner beziehen sich auf vordefinierte Windows-Symbole. Die Symbole werden mit der Funktion *LoadIcon* geladen.

idi_Application	Voreingestelltes Anwendungssymbol.
idi_Asterisk	Ein »i« in einem Kreis (verwendet bei Informationsmeldungen).
idi_Exclamation	Ausrufezeichen (verwendet bei Warnungsmeldungen).
idi_Hand	Symbol in Handform (verwendet bei ernsten Warnungsmeldungen).
idi_Question	Fragezeichen (verwendet bei Eingabemeldungen).

ie_Xxxx

Diese Flags sind negative Werte, die einen Fehler beim Öffnen einer Schnittstelle anzeigen. Sie werden von der Funktion *OpenComm* zurückgegeben.

ie_BadID	Ungültige oder nicht unterstützte ID.
ie_BaudRate	Nicht unterstützte Baudrate.
ie_ByteSize	Ungültige Bytegröße.
ie_Default	Fehler in Standardparametern.
ie_Hardware	Hardware nicht vorhanden.
ie_Memory	Es können keine Warteschlangen eingerichtet werden.
ie_NOpen	Gerät nicht bereit.
ie_Open	Gerät bereits aktiviert.

lb_Xxxx

Diese Werte werden von Windows-Listenbotschaften (z.B. *lb_AddString*) zurückgegeben. Ein negativer Wert zeigt einen Fehler an.

lb_Err	Ein Fehler ist aufgetreten. Die Aktion wurde nicht durchgeführt.
lb_ErrSpace	Der Platz in der Liste reicht zum Durchführen der Aktion nicht aus.
lb_Okay	Kein Fehler.

lbn_Xxxx

Diese Benachrichtigungscodes gelten nur für Listen des Stils *lbs_Notify*. Sie werden in *wm_Command*-Botschaften übergeben und zeigen eine Aktion an.

lbn_DblClk	Wird übergeben, wenn der Anwender einen String durch einen Doppelklick auswählt.
lbn_ErrSpace	Zeigt an, daß im System kein Speicher mehr frei ist.
lbn_KillFocus	Zeigt an, daß die Liste den Fokus verloren hat.
lbn_SelChange	Wird bei Änderung der Auswahl übergeben.
lbn_SetFocus	Zeigt an, daß die Liste den Fokus erhalten hat.

lbs_Xxxx

Diese Konstanten werden verwendet, um den Stil einer Liste festzulegen, wenn sie mit einer der Funktionen *CreateWindow* oder *CreateWindowEx* erzeugt wird.

lbs_ExtendedSel	Der Anwender kann mehrere Einträge auswählen, indem er die Shift -Taste und die Maus benutzt, oder spezielle Tastenkombinationen verwendet.

lbs_HasStrings	Bestimmt eine besitzergezeichnete Liste, die Einträge enthält, die aus Strings bestehen. Die Liste behält den Speicher und die Zeiger für die Strings bei, so daß die Anwendung die Botschaft *lb_GetText* verwenden kann, um den Text für einen bestimmten Eintrag abzurufen.
lbs_MultiColumn	Bestimmt eine mehrspaltige Liste, die horizontal gerollt werden kann. Die Botschaft *lb_SetColumnWidth* legt die Breite der Spalten fest.
lbs_MultipleSel	Die Stringauswahl wechselt jedesmal ihren Zustand, wenn der Anwender den String einfach oder doppelt anklickt. Jede Anzahl von Strings kann ausgewählt werden.
lbs_NoIntegralHeight	Die Liste hat genau die Größe, die von der Anwendung angegeben wurde, als sie die Liste erzeugt hat. Normalerweise legt Windows die Größe einer Liste so fest, daß die Liste keine Teileinträge anzeigt.
lbs_NoRedraw	Die Anzeige der Liste wird nicht aktualisiert, wenn Änderungen gemacht werden. Dieser Stil kann jederzeit geändert werden, indem eine Botschaft *wm_SetRedraw* übergeben wird.
lbs_Notify	Das übergeordnete Fenster empfängt immer dann eine Eingabebotschaft, wenn der Anwender einen String einfach oder doppelt anklickt.
lbs_OwnerDrawFixed	Der Besitzer der Liste ist für das Zeichnen ihres Inhalts verantwortlich; die Einträge in der Liste haben alle die gleiche Höhe.
lbs_OwnerDrawVariable	Der Besitzer der Liste ist für das Zeichnen ihres Inhalts verantwortlich; die Einträge in der Liste haben eine variable Höhe.
lbs_Sort	Strings in der Liste werden alphabetisch sortiert.
lbs_Standard	Strings in der Liste werden alphabetisch sortiert und das übergeordnete Fenster empfängt immer dann eine Eingabebotschaft, wenn der Anwender einen String einfach oder doppelt anklickt. Die Liste ist umrandet.
lbs_UseTabStops	Erlaubt einer Liste Tabulatorzeichen zu erkennen und zu erweitern, wenn sie ihre Strings zeichnet. Die voreingestellten Tabulatorpositionen sind 32 Dialogeinheiten. Eine Dialogeinheit ist eine horizontale oder vertikale Distanz. Eine horizontale Dialogeinheit ist gleich 1/4 der zugehörigen Basiseinheit. Die Basiseinheiten werden entsprechend der Höhe und Breite der momentanen Systemschrift berechnet. Die Funk-

tion *GetDialogBaseUnits* liefert die gegenwärtigen Dialogbasiseinheiten in Pixel zurück.

lbs_WantKeyboardInput Der Besitzer der Liste erhält immer *wm_VKeyToItem*- oder *wm_CharToItem*-Botschaften, wenn der Anwender eine Taste drückt, während die Liste den Fokus besitzt. Dies erlaubt einer Anwendung, eine spezielle Bearbeitung der Tastatureingabe vorzunehmen.

lc_Xxxx

Diese Konstanten repräsentieren die Linienzeichnungsfähigkeiten eines Geräts.

lc_None Keine Linienzeichnungsfähigkeiten.

lc_Interiors Gerät kann Ausfüllungen durchführen.

lc_Marker Gerät kann Markierungssymbole darstellen.

lc_PolyLine Gerät kann Linenzüge darstellen.

lc_PolyMarker Gerät kann Markierungssybole und Linienzüge darstellen.

lc_Styled Gerät kann verschiedene Linien darstellen.

lc_Wide Gerät kann breite Linien darstellen.

lc_WideStyled Gerät kann verschiedene breite Linien darstellen.

lf_FaceSize

Die Konstante *lf_FaceSize* gibt die Anzahl der Array-Elemente (*Char*) an, die zum Speichern des Schriftbildnamens im Feld *lfFaceName* eines *TLogFont*-Record bereitgestellt werden. Der Vorgabewert ist 32.

lmem_Xxxx

Diese Flags bestimmen Eigenschaften des lokalen Speicherblocks, der mit den Funktionen *LocalAlloc* oder *LocalReAlloc* erzeugt wurde. *lmem_Discardable*, *lmem-Discarded* und *lmem_LockCount* werden mit der Funktion *LocalFlags* verwendet.

lmem_Discardable Reserviert verwerfbaren Speiche und kann nur mit *lmem_Moveable* benutzt werden.

lmem_Discarded Block wurde verworfen (nur *LocalFlags*).

lmem_Fixed Reserviert festen Speicher.

lmem_LockCount Wird dieser Wert mit dem niederwertigen Byte des Rückgabewerts der Funktion *LocalFlags* kombiniert, wird der Referenzzähler des Speicherblocks zurückgegeben (nur *LocalFlags*).

lmem_Modify	Modifiziert das Flag *lmem_Discardable*. Kann nur mit *lmem_Discardable* benutzt werden.
lmem_Moveable	Reserviert verschiebbaren Speicher. Kann nicht mit *lmem_Fixed* benutzt werden.
lmem_NoCompact	Verdichtet oder verwirft keinen Speicher, um die gewünschte Reservierung durchzuführen.
lmem_NoDiscard	Verwirft keinen Speicher, um die gewünschte Reservierung durchzuführen.
lmem_ZeroInit	Initialisiert den Speicherinhalt auf 0.

LPTx

LPTx ist eine Bitmaske für das Feld *ID* einer *TDCB*-Struktur. Ist die Bitmaske gesetzt, ist das Gerät an einer parallelen Schnittstelle angeschlossen.

ma_Xxxx

Die Codes werden von einer *wm_MouseActivate*-Botschaft zurückgegeben und zeigen an, ob das Fenster aktiviert und das Mausereignis verworfen werden soll.

ma_Activate	Fenster aktivieren.
ma_ActivateAndEat	Fenster aktivieren und Mausereignis verwerfen.
ma_NoActivate	Fenster nicht aktivieren.

mb_Xxxx

Diese Flags bestimmen die Eigenschaften eines Meldungsfensters, das mit der Funktion *MessageBox* erzeugt wurde – sie können kombiniert werden.

mb_AbortRetryIgnore	Das Meldungsfenster enthält drei Schalter: *Beenden*, *Wiederholen* und *Ignorieren*.
mb_ApplModal	Der Anwender muß auf das Meldungsfenster reagieren, bevor er die Arbeit in dem Fenster fortsetzt, das der Parameter *Parent* bezeichnet. Der Anwender kann sich jedoch zu den Fenstern anderer Anwendungen bewegen und in diesen Fenstern arbeiten. *mb_ApplModal* ist der Standardwert, wenn weder *mb_SystemModal* noch *mb_TaskModal* angegeben ist.
mb_DefButton1	Erster Schalter ist Voreinstellung. Man beachte, daß der erste Schalter immer die Voreinstellung ist, wenn nicht *mb_DefButton2* oder *mb_DefButton3* angegeben ist.
mb_DefButton2	Zweiter Schalter ist Voreinstellung.

mb_DerButton3	Dritter Schalter ist Voreinstellung.
mb_IconAsterisk	Gleichbedeutend mit *mb_IconInformation*.
mb_IconExclamation	Ein Ausrufezeichensymbol erscheint im Meldungsfenster.
mb_IconHand	Gleichbedeutend mit *mb_IconStop*.
mb_IconInformation	Ein Symbol, das aus einem kleinen »i« in einem Kreis besteht, erscheint im Meldungsfenster.
mb_IconQuestion	Ein Fragezeichensymbol erscheint im Meldungsfenster.
mb_IconStop	Ein Stopzeichensymbol erscheint im Meldungsfenster.
mb_Ok	Das Meldungsfenster enthält einen Schalter: *OK*.
mb_OkCancel	Das Meldungsfenster enthält zwei Schalter: *OK* und *Abbrechen*.
mb_RetryCancel	Das Meldungsfenster enthält zwei Schalter: *Abbrechen* und *Wiederholen*.
mb_SystemModal	Alle Anwendungen werden unterbrochen, bis der Anwender dem Meldungsfenster antwortet. Wenn die Anwendung nicht *mb_IconHand* vorgibt, wird das Meldungsfenster erst modal, nachdem es erstellt worden ist; folglich empfangen das übergeordnete und andere Fenster weiterhin Botschaften, die sich aus seiner Aktivierung ergeben. Systemmodale Meldungsfenster werden verwendet, um den Benutzer von ernsten, möglicherweise schadensverursachenden Fehlern zu benachrichtigen, die sofortige Aufmerksamkeit erfordern (z.B. Speicherüberlauf).
mb_TaskModal	Gleichbedeutend mit *mb_ApplModal*, jedoch mit der Ausnahme, daß alle Hauptfenster, die zu der aktuellen Task gehören, gesperrt werden, wenn der Parameter *WndParent* 0 ist. Dieses Flag sollte dann benutzt werden, wenn die aufrufende Anwendung oder Bibliothek kein verfügbares Fenster-Handle hat, aber eines benötigt, um Eingaben in andere Fenster der aktuellen Anwendung zu verhindern, ohne andere Anwendungen zu unterbrechen.
mb_YesNo	Meldungsfenster enthält zwei Schalter: *Ja* und *Nein*.
mb_YesNoCancel	Meldungsfenster enthält drei Schalter: *Ja*, *Nein* und *Abbrechen*.

Für bestimmte Gruppen von *mb_*Konstanten sind verschiedene Bitmasken definiert.

Maske	Enthaltene Konstanten
mb_DefMask	*mb_DefButton1*, *mb_DefButton2*, *mb_DefButton3*
mb_IconMask	*mb_IconAsterisk*, *mb_IconExclamation*, *mb_IconHand*, *mb_IconInformation*, *mb_IconQuestion*, *mb_IconStop*
mb_ModeMask	*mb_ApplModal*, *mb_SystemModal*, *mb_TaskModal*
mb_TypeMask	*mb_AbortRetryIgnore*, *mb_Ok*, *mb_OkCancel*, *mb_RetryCancel*, *mb_YesNo*, *mb_YesNoCancel*

meta_Xxxx

Diese Konstanten entsprechen einzelnen Funktionen der grafischen Geräteschnittstelle (GDI). Der Wert von *meta_Arc* ist also der numerische Index der GDI-Funktion *Arc*. Die Indexwerte dieser Funktionen und die entsprechenden Konstanten können in einem Metafile gespeichert werden. Dieses Metafile ist eine Liste von GDI-Befehlen, die von einer Anwendung zur Erzeugung grafischer Ausgaben abgespielt werden kann.

meta_AnimatePalette
meta_Arc
meta_BitBlt
meta_Chord
meta_CreateBitmap
meta_CreateBitmapIndirect
meta_CreateBrush
meta_CreateBrushIndirect
meta_CreateFontIndirect
meta_CreatePalette
meta_CreatePatternBrush
meta_CreatePenIndirect
meta_CreateRegion
meta_DeleteObject
meta_DIBBitBlt
meta_DIBCreatePatternBrush
meta_DIBStretchBlt
meta_DrawText
meta_Ellipse
meta_Escape
meta_ExcludeClipRect
meta_ExtTextOut
meta_FillRegion
meta_FloodFill
meta_FrameRegion
meta_IntersectClipRect
meta_InvertRegion
meta_LineTo
meta_MoveTo
meta_OffsetClipRgn
meta_OffsetViewportOrg
meta_OffsetWindowOrg
meta_PaintRegion
meta_PatBlt
meta_Pie
meta_Polygon
meta_PolyLine
meta_PolyPolygon
meta_RealizePalette
meta_Rectangle
meta_ResizePalette
meta_RestoreDC
meta_RoundRect
meta_SaveDC
meta_ScaleViewportExt
meta_ScaleWindowExt
meta_SelectClipRegion
meta_SelectObject
meta_SelectPalette
meta_SetBKColor
meta_SetBKMode
meta_SetDIBToDEV
meta_SetMapMode
meta_SetMapperFlags
meta_SetPalEntries
meta_SetPixel
meta_SetPolyFillMode
meta_SetRelAbs
meta_SetROP2
meta_SetStretchBltMode

meta_SetTextAlign
meta_SetTextCharExtra
meta_SetTextColor
meta_SetTextJustification
meta_SetViewportExt
meta_SetViewportOrg
meta_SetWindowExt
meta_SetWindowOrg
meta_StretchBlt
meta_TextOut

mf_Xxxx

Die folgenden Konstanten werden als Flags in vielen Menü-Funktionen und in der Botschaft *wm_MenuSelect* verwendet.

mf_Bitmap Benutzt ein Bitmap als Menüoption.

mf_ByCommand Gibt die Befehls-ID der vorhandenen Menüoption an.

mf_ByPosition Gibt die Position einer existierenden Menüoption an. Die erste Option im Menü ist an Position 0.

mf_Checked Setzt eine Auswahlmarkierung neben die Menüoption. Wenn die Anwendung Bitmaps für Auswahlmarkierungen zur Verfügung gestellt hat (siehe auch *SetMenuItemBitmaps*), bewirkt das Setzen dieses Flags die Anzeige des Bitmaps »Ausgewählt« neben der Menüoption.

mf_Disabled Sperrt die Menüoption, so daß sie nicht ausgewählt werden kann, versetzt sie aber nicht in den nichtaktivierbaren Zustand.

mf_Enabled Schaltet die Menüoption ein, so daß sie ausgewählt werden kann und stellt sie aus dem nichtaktivierbaren Zustand wieder her.

mf_Grayed Sperrt die Menüoption, so daß sie nicht ausgewählt werden kann und versetzt sie in den nichtaktivierbaren Zustand.

mf_Help Zeigt an, daß der Eintrag ein Hilfe-Menüeintrag ist.

mf_MenuBarBreak Gleichbedeutend mit *mf_MenuBreak* mit der Ausnahme, daß bei Pop-Up-Menüs die neue Spalte von der alten durch eine vertikale Linie getrennt wird.

mf_MenuBreak Setzt die Menüoption auf eine neue Zeile für statische Menüleistenelemente. Bei Pop-Up-Menüs setzt dieses Flag das Element in eine neue Spalte, ohne Trennungslinie zwischen den Spalten.

mf_MouseSelect Zeigt an, daß der Eintrag mit der Maus ausgewählt wurde.

mf_OwnerDraw Gibt an, daß die Menüoption eine besitzergezeichnete Option ist. Das Fenster, das das Menü besitzt, emp-

	fängt eine Botschaft *wm_MeasureItem*, wenn das Menü das erste Mal dargestellt wird, um Höhe und Breite der Menüoption abzurufen. Die Botschaft *wm_DrawItem* wird immer dann gesendet, wenn die Anwendung das Erscheinungsbild der Menüoption aktualisieren muß. Diese Einstellung des Parameters gilt nicht für Hauptmenüoptionen.
mf_Popup	Gibt an, daß der Option ein Pop-Up-Menü zugeordnet ist. Dieses Flag sollte benutzt werden, um entweder ein Haupt-Pop-Up-Menü einzufügen oder ein hierarchisches Pop-Up-Menü an eine Pop-Up-Menüoption anzufügen.
mf_Separator	Zeichnet eine horizontale Trennungslinie. Man kann dieses Flag nur in einem Pop-Up-Menü benutzen. Diese Linie kann nicht in den nichtaktivierbaren Zustand versetzt, gesperrt oder hervorgehoben werden.
mf_String	Gibt an, daß die Menüoption ein String ist.
mf_SysMenu	Zeigt an, daß sich der Eintrag im Systemmenü befindet.
mf_Unchecked	Setzt keine Auswahlmarkierung neben die Menüoption. Keine Auswahl ist die Voreinstellung, wenn weder *mf_Checked* noch *mf_Unchecked* gesetzt ist. Wenn die Anwendung Bitmaps für Auswahlmarkierungen zur Verfügung gestellt hat (siehe *SetMenuItemBitmaps*), bewirkt das Setzen dieses Flags, daß das Bitmap »Nicht ausgewählt« neben der Menüoption dargestellt wird.
mf_Unhilite	Entfernt die Hervorhebung vom Menüeintrag.

mk_Xxxx

Diese Masken werden kombiniert und in Mausklickbotschaften übergeben, um den Status bestimmter Tasten und Maustasten zu ermitteln.

mk_Control	Gesetzt, wenn die Ctrl -Taste gedrückt wird.
mk_LButton	Gesetzt, wenn die linke Maustaste gedrückt wird.
mk_MButton	Gesetzt, wenn die mittlere Maustaste gedrückt wird.
mk_RButton	Gesetzt, wenn die rechte Maustaste gedrückt wird.
mk_Shift	Gesetzt, wenn die Shift -Taste gedrückt wird.

mm_Xxxx

Diese Konstanten bestimmen den Abbildungsmodus des Gerätekontextes. Dieser Modus bestimmt die Konvertierung virtueller Einheiten in Geräteeinheiten und die Ausrichtung der Achsen. Die Konstanten werden in den Funktionen *GetMapMode* und *SetMapMode* verwendet.

mm_Anisotropic Für Anwendungen, die eine virtuelle Einheit auf eine beliebige physikalische Einheit abbilden. Die x- und y-Achsen können beliebig skaliert werden.

mm_HiEnglish Für Anwendungen, die eine virtuelle Einheit auf 0.001 Zoll abbilden. Positive y-Richtung nach oben.

mm_HiMetric Für Anwendungen, die eine virtuelle Einheit auf 0.01 Millimeter abbilden. Positive y-Richtung nach oben.

mm_Isotropic Für Anwendungen, die eine virtuelle Einheit auf eine beliebige physikalische Einheit abbilden. Eine x-Achseneinheit ist gleich einer y-Achseneinheit.

mm_LoEnglish Für Anwendungen, die eine virtuelle Einheit auf 0.01 Zoll abbilden. Positive y-Richtung nach oben.

mm_LoMetric Für Anwendungen, die eine virtuelle Einheit auf 0.1 Millimeter abbilden. Positive y-Richtung nach oben.

mm_Text Für Anwendungen, die eine virtuelle Einheit auf ein Pixel abbilden. Positive y-Richtung nach unten.

mm_TWIPS Für Anwendungen, die eine virtuelle Einheit auf 1/1440 Zoll abbilden. Positive y-Richtung nach oben.

msgf_Xxxx

Diese Codes werden im Parameter *Code* der beiden Botschaftsfilterfunktionen *wh_MsgFilter* und *wh_SysMsgFilter* übergeben. Sie bestimmen den Typ der zu bearbeitenden Botschaften.

	Zu bearbeitende Botschaften *wh_MsgFilter*	Konstante *wh_SysMsgFilter*
msgf_DialogBox	Meldungs- und Dialogfenster	Nur Dialogfenster
msgf_Menu	Tastatur- und Mausbotschaften	Tastatur- und Mausbotschaften
msgf_MessageBox	Nur Meldungsfensterbotschaften	

obj_Xxxx

Diese Konstanten bestimmen den Typ des GDI-Objekts (Stift oder Pinsel), das in einem Aufruf der Funktion *EnumObjects* aufgelistet werden soll, und sollten als Parameter *ObjectType* der Funktion *EnumObjects* übergeben werden.

obj_Pen	Listet Stifte
obj_Brush	Listet Pinsel

obm_Xxxx

Diese Konstanten bestimmen die Bitmaps, die unter Windows für die Verwendung durch das System vordefiniert sind. Programmierer können mit der Funktion *LoadBitmaps* auf diese Bitmaps zugreifen. Konstanten, die mit *obm_Old_* beginnen, beziehen sich auf Bitmaps, die in Windows-Versionen vor 3.0 verwendet wurden.

obm_BTNCorners
obm_BTSize
obm_Check
obm_CheckBoxes
obm_Close
obm_Combo
obm_DnArrow
obm_DnArrowD
obm_LfArrow
obm_LfArrowD
obm_MnArrow
obm_Old_Close
obm_Old_DnArrow
obm_Old_LfArrow
obm_Old_Reduce
obm_Old_Restore
obm_Old_RgArrow
obm_Old_UpArrow
obm_Old_Zoom
obm_Reduce
obm_Reduced
obm_Restore
obm_Restored
obm_RgArrow
obm_RgArrowD
obm_Size
obm_UpArrow
obm_UpArrowD
obm_Zoom
obm_ZoomD

oda_Xxxx

Diese Konstanten definieren die Aktion, die für ein besitzergezeichnetes Steuerelement durchgeführt werden soll. Sie werden im Feld *itemAction* der Struktur *TDrawItemStruct* verwendet. Das Feld kann auch eine Kombination dieser Konstanten aufnehmen.

oda_DrawEntire	Das ganze Steuerelement muß gezeichnet werden.
oda_Focus	Der Fokus wurde empfangen oder abgegeben.
oda_Select	Der Auswahlstatus wurde verändert.

ods_Xxxx

Diese Konstanten definieren den Status eines besitzergezeichneten Steuerelements nach seiner Zeichnung. Sie werden im Feld *itemState* der Struktur *TDrawItemStruct* verwendet. Das Feld kann auch eine Kombination dieser Werte aufnehmen.

ods_Checked	Menüeintrag markieren.

ods_Disabled	Menüeintrag aktivieren.
ods_Focus	Der Menüeintrag erhält den Fokus.
ods_Grayed	Menüeintrag wird in den nichtaktivierbaren Zustand versetzt.
ods_Selected	Menüeintrag auswählen.

odt_Xxxx

Diese Konstanten bestimmen den Typ des besitzergezeichneten Steuerelements. Sie werden im Feld *CtlType* der Strukturen *TCompareItemStruct, TDrawItemStruct* und *TMeasureItemStruct* verwendet.

odt_Menu	Besitzergezeichnetes Menü.
odt_ListBox	Besitzergezeichnete Liste.
odt_ComboBox	Besitzergezeichnetes Kombinationsfenster.
odt_Button	Besitzergezeichneter Schalter.

of_Xxxx

Diese Konstanten werden kombiniert und in Aufrufen der Funktionen *OpenFile* oder *_lopen* übergeben. Sie bestimmen die Art der Dateiöffnung..

of_Cancel	Fügt einen Aktionsschalter für Abbruch in das Dialogfenster *of_Prompt* ein. Die Betätigung des Abbruchschalters weist *OpenFile* an, eine Fehlermeldung `Datei nicht gefunden` zurückzugeben (nur *OpenFile*).
of_Create	Weist *OpenFile* an, eine neue Datei zu erstellen. Falls die Datei schon vorhanden ist, wird sie auf die Länge 0 gekürzt (nur *OpenFile*).
of_Delete	Löscht die Datei (nur *OpenFile*).
of_Exist	Öffnet die Datei und schließt sie wieder. Wird für eine Prüfung auf die Existenz einer Datei verwendet (nur *OpenFile*).
of_Parse	Füllt die Datenstruktur *TOFStruct*, die im Parameter *ReOpenBuff* übergeben wird. Weitere Aktionen finden nicht statt.
of_Prompt	Zeigt ein Dialogfenster an, wenn die angesprochene Datei nicht existiert. Das Dialogfenster informiert den Benutzer, daß Windows die Datei nicht finden kann und fordert den Benutzer auf, eine Diskette mit der Datei in Laufwerk A einzulegen (nur *OpenFile*).
of_Read	Öffnet die Datei nur zum Lesen.

of_ReadWrite	Öffnet die Datei zum Lesen und Schreiben.
of_Reopen	Öffnet die Datei unter Verwendung der Informationen in der durch *ReOpenBuff* bezeichneten Struktur *TOFStruct* (nur *OpenFile*).
of_Share_Compat	Öffnet eine Datei im Kompatibilitätsmodus und ermöglicht dadurch das Öffnen der Datei, sooft es nötig ist. Die Funktion kann nicht ausgeführt werden, wenn die Datei in einem anderen Modus für gemeinsame Nutzung (*Share*) geöffnet wurde.
of_Share_Deny_None	Öffnet die Datei, ohne anderen Prozessen den Lese- oder Schreibzugriff auf diese Datei zu verweigern. Die Funktion kann nicht ausgeführt werden, wenn die Datei von einem anderen Prozeß im Kompatibilitätsmodus geöffnet wurde.
of_Share_Deny_Read	Öffnet die Datei und verweigert anderen Prozessen den Lesezugriff auf die Datei. Die Funktion kann nicht ausgeführt werden, wenn die Datei im Kompatibilitätsmodus oder für Lesezugriffe von einem anderen Prozeß geöffnet wurde.
of_Share_Deny_Write	Öffnet die Datei und verweigert anderen Prozessen Schreibzugriffe auf die Datei. Die Funktion kann nicht ausgeführt werden, wenn die Datei im Kompatibilitätsmodus oder für Schreibzugriff von einem anderen Prozeß geöffnet wurde.
of_Share_Exclusive	Öffnet die Datei im Exklusivmodus und verweigert anderen Prozessen sowohl Lese- als auch Schreibzugriffe auf die Datei. Die Funktion kann nicht ausgeführt werden, wenn die Datei in irgendeinem anderen Modus für Lese- oder Schreibzugriffe geöffnet wurde. Das gilt auch für den aktuellen Prozeß.
of_Verify	Überprüft, ob Datum und Zeit der Datei dieselben sind, wie beim vorherigen Öffnen. Dieser Wert ist zur zusätzlichen Überprüfung von schreibgeschützten Dateien nützlich (nur *OpenFile*).
of_Write	Öffnet die Datei nur zum Schreiben.

Out_Xxxx

Diese Konstanten bestimmen die Ausgabepräzision von Schriften, die mit der Funktion *CreateFont* erzeugt werden.

Out_Character_Precis	*Out_String_Precis*
Out_Default_Precis	*Out_Stroke_Precis*

pc_Xxxx

Diese Flags bestimmen Informationen über eine Palette. Sie werden im Feld *peFlags* der Struktur *TPaletteEntry* verwendet und in Aufrufen einer der Funktionen *AnimatePalette*, *GetPaletteEntries*, *GetSystemPaletteEntries* und *SetPaletteEntries* übergeben.

pc_Explicit — Bestimmt, daß das niederwertige Word des virtuellen Paletteneintrags einen Hardware-Palettenindex beschreibt. Dieses Flag erlaubt einer Anwendung, den Inhalt der Palette eines Anzeigegeräts anzuzeigen.

pc_NoCollapse — Bestimmt, daß die Farbe in einem ungenutzten Eintrag in der Systempalette abgelegt wird, anstelle der Anpassung an einen existierenden Farbeintrag in der Systempalette. Sind keine ungenutzten Einträge in der Systempalette vorhanden, wird die Farbe normal angepaßt.
Ist diese Farbe erst in der Systempalette eingetragen ist, können Farben in anderen virtuellen Paletten dieser Farbe angepaßt werden.

pc_Reserved — Bestimmt, daß der virtuelle Paletteneintrag für Palettenanimation genutzt wird. Dadurch wird verhindert, daß andere Fenster die Farben an diesen Paletteneintrag anpassen, da sich die Farbe zeitweilig ändert. Wenn ein ungenutzter Systempaletteneintrag verfügbar ist, wird diese Farbe in den Eintrag plaziert, andernfalls steht diese Farbe für Animation nicht zur Verfügung.

Diese Konstanten repräsentieren die Polygonzeichnungsfähigkeiten eines Geräts.

pc_Interiors — Gerät kann Ausfüllungen durchführen.

pc_None — Gerät unterstützt die Darstellung von Polygonen nicht.

pc_Polygon — Gerät kann Polygone darstellen.

pc_Rectangle — Gerät kann Rechtecke darstellen.

pc_ScanLine — Gerät kann Rasterzeilen darstellen.

pc_Styled — Gerät kann verschiedene Rahmen darstellen.

pc_Trapezoid — Gerät kann Trapeze darstellen.

pc_Wide — Gerät kann breite Rahmen darstellen.

pc_WideStyled — Gerät kann verschiedene breite Rahmen darstellen.

pc_WindPolygon — Gerät kann überschneidende Polygone darstellen.

pm_Xxxx

Diese Optionen bestimmen den Umgang mit Botschaften, die von der Funktion *PeekMessage* verarbeitet werden.

pm_NoRemove	Botschaften werden nach der Bearbeitung durch *PeekMessage* nicht aus der Warteschlange gelöscht.
pm_NoYield	Hindert die aktuelle Task daran, zu unterbrechen und Systemressourcen an eine andere Task abzugeben.
pm_Remove	Botschaften werden nach der Bearbeitung durch *PeekMessage* aus der Warteschlange gelöscht.

Polygonfüll-Modi

Diese Konstanten bestimmen die Methode, nach der Polygone mit mehr als einer assoziierten Region gefüllt werden. Sie werden in Aufrufen einer der Funktionen *CreatePolygonRgn*, *CreatePolyPolygonRgn*, *GetPolyFillMode* und *SetPolyFillMode* verwendet.

Alternate	Regionen werden alternierend ausgefüllt.
Winding	Jede Region wird gefüllt.

pr_JobStatus

pr_JobStatus wird als Parameter *wParam* in der Botschaft *wm_SpoolerStatus* übergeben.

proc_Xxxx

Diese Konstanten werden als Parameter *ProcName* der Funktion *GetProcAdress* übergeben, wenn das Modul-Handle im Parameter *Module* einen Gerätetreiber bezeichnet.

proc_ExtDeviceMode
proc_DeviceCapabilities
proc_OldDeviceMode

ps_Xxxx

Die folgenden Konstanten legen Stiftstile fest. Sie werden in der Funktion *CreatePen* verwendet.

ps_Solid	_____
ps_Dash	- - - - -

ps_Dot	. . . . .
ps_DashDot	- · - · - · -
ps_DashDotDot	- ·· - ·· -
ps_Null	Unsichtbar
ps_InsideFrame	Wird zum Zeichnen des inneren Rahmens von Polygonen und Linienzügen verwendet.

r2_Xxxx

Diese Konstanten bestimmen den Zeichenmodus, um die Farben des aktuellen Stifts mit den existierenden Farben zu kombinieren. Sie werden als Parameter in den Funktionen *GetROP2* und *SetROP2* übergeben.

r2_Black	Pixel ist immer schwarz.
r2_CopyPen	Pixel hat die Stiftfarbe.
r2_MaskNotPen	Pixel ist eine Kombination der Farben, die dem Bildschirm und der inversen Stiftfarbe gemeinsam sind.
r2_MaskPen	Pixel ist eine Kombination der Farben, die dem Stift und dem Bildschirm gemeinsam sind.
r2_MaskPenNot	Pixel ist eine Kombination der Farben, die dem Stift und der inversen Bildschirmfarbe gemeinsam sind.
r2_MergeNotPen	Pixel ist eine Kombination der Bildschirmfarbe und der inversen Stiftfarbe.
r2_MergePen	Pixel ist eine Kombination der Stiftfarbe und der Bildschirmfarbe.
r2_MergePenNot	Pixel ist eine Kombination von Stiftfarbe und der inversen Bildschirmfarbe.
r2_NOP	Pixel bleibt unverändert.
r2_Not	Pixel ist invers zur Bildschirmfarbe.
r2_NotCopyPen	Pixel hat die inverse Stiftfarbe.
r2_NotMaskPen	Pixel ist die Invertierung der Farbe *r2_MaskPen*.
r2_NotMergePen	Pixel ist die Invertierung der Farbe *r2_MergePen*.
r2_NotXORPen	Pixel ist die Invertierung der Farbe *r2_XorPen*.
r2_White	Pixel ist immer weiß.
r2_XORPen	Pixel ist eine Kombination der Farben im Stift und auf dem Bildschirm, die nicht in beiden gleichzeitig vorkommen.

rc_Xxxx

Diese Konstanten repräsentieren die Rasterfähigkeiten eines Geräts.

rc_Banding	Benötigt Frequenzunterstützung.
rc_BigFont	Unterstützt Schriften mit mehr als 64 KByte.
rc_BitBlt	Fähig zur Bitmap-Übertragung.
rc_Bitmap64	Fähig zur Unterstützung von Bitmaps mit mehr als 64 KByte.
rc_Di_Bitmap	Fähig zur Unterstützung von *SetDIBits* und *GetDIBits*.
rc_DIBToDev	Fähig zur Unterstützung von *SetDIBitsToDevice*.
rc_FloodFill	Fähig zum Ausmalen.
rc_GDI20_Output	Fähig zur Unterstützung von Merkmalen der Version Windows 2.0.
rc_Palette	Auf Farbpalette aufbauendes Gerät.
rc_Scaling	Fähigkeit zum Skalieren.
rc_StretchBlt	Fähig zur Durchführung von *StretchBlt*.
rc_StretchDIB	Fähig zur Durchführung von *StretchDIBits*.

Region-Flags

Diese Flags bestimmen den Typ einer grafischen Region. Sie werden als Rückgabewerte der Funktionen *CombineRgn, ExcludeClipRect, ExcludeUpdateRgn, GetClipBox, GetRgnBox, GetUpdateRgn, OffsetClipRgn, OffsetRgn* und *SelectClipRgn* verwendet.

ComplexRegion	Die neue Region hat überlappende Ränder.
Error	Keine neue Region erzeugt.
NullRegion	Die neue Region ist leer.
SimpleRegion	Die neue Region hat keine überlappenden Ränder.

Ressourcentyp-Konstante

Dieser Wert ist die Anzahl der definierten Ressourcentypen minus 1. Fügen Sie einen neuen Ressourcentyp hinzu, müssen Sie *Difference* entsprechend erhöhen. Der Vorgabewert für *Difference* ist 11.

rgn_Xxxx

Diese Flags bestimmen die Kombinationsmethode, die von der Funktion *CombineRgn* zur Kombination zweier Regionen verwendet wird.

rgn_And	Benutzt überlappende Bereiche beider Regionen (Schnittpunkt).
rgn_Copy	Erzeugt eine Kopie von Region 1 (wird durch *SrcRgn1* bezeichnet).
rgn_Diff	Speichert die Bereiche der durch den Parameter *SrcRgn1* bezeichneten Region 1, die nicht Teil der durch *SrcRgn2* bezeichneten Region 2 sind.
rgn_Or	Verbindet beide Regionen (Vereinigung).
rgn_Xor	Verbindet beide Regionen, entfernt aber überlappende Bereiche.

rt_Xxxx

Diese Konstanten repräsentieren die verfügbaren Typen von Windows-Ressourcen. Sie werden als Parameter der Funktion *FindResource* verwendet.

rt_Accelerator	Akzelerator-Tabelle.
rt_Bitmap	Bitmap-Ressource.
rt_Cursor	Zeiger-Ressource.
rt_Dialog	Dialogfenstervorlage.
rt_Font	Schriftressource.
rt_FontDir	Schriftenverzeichnisressource.
rt_Icon	Symbol-Ressource.
rt_Menu	Menüressource.
rt_RCData	Benutzerdefinierte Ressource (Rohdaten).
rt_String	String-Ressource.

s_Xxxx

Diese Konstanten werden in verschiedenen Sound-Funktionen verwendet. Sie werden entsprechend der Funktionen aufgelistet, in denen sie verwendet werden. Folgende negativen Fehlerwerte werden von Sound-Funktionen zurückgegeben.

s_SerBDNT	Ungültige Note.
s_SerDCC	Ungültiger Notenzähler.
s_SerDDR	Ungültige Dauer.
s_SerDFQ	Ungültige Frequenz.
s_SerDLN	Ungültige Notenlänge.

s_SerDMD	Ungültiger Modus.
s_SerDPT	Ungültiger Abstand.
s_SerDSH	Ungültige Form.
s_SerDSR	Ungültige Quelle.
s_SerDST	Ungültiger Status.
s_SerDTP	Ungültiges Tempo.
s_SerDVL	Ungültige Lautstärke.
s_SerDVNA	Gerät ist nicht verfügbar.
s_SerMACT	Musik ist aktiv.
s_SerOFM	Speicherüberlauf.
s_SerQFUL	Warteschlange ist voll.

SetSoundNoise: Diese Konstanten bestimmen die Tonquelle. *N* ist ein Wert zur Ermittlung der Zielfrequenz.

s_Period512	Quellfrequenz ist N/512 (hoher Ton); weniger heller Ton.
s_Period1024	Quellfrequenz ist N/1024.
s_Period2048	Quellfrequenz ist N/2048 (tiefer Ton); hellerer Ton.
s_PeriodVoice	Quellfrequenz von Voice-Kanal 3.
s_White512	Quellfrequenz ist N/512 (hoher Ton); weniger heller Ton.
s_White1024	Quellfrequenz ist N/1024.
s_White2048	Quellfrequenz ist N/2048 (tiefer Ton); hellerer Ton.
s_WhiteVoice	Quellfrequenz von Voice-Kanal 3.

SetVoiceAccent: Diese Konstanten bestimmen, wie der Akzent der Noten gespielt werden soll.

s_Legato	Der Ton wird für die ganze Dauer gehalten und mit dem Beginn des nächsten Tons ausgeblendet.
s_Normal	Der Ton wird für die ganze Dauer gehalten und bricht ab, bevor der nächste Ton beginnt.
s_Staccato	Der Ton wird nur für einen Teil der Dauer gehalten und weist einen deutlichen Abstand zwischen ihm und dem nächsten Ton auf.

WaitSoundState: Diese Konstanten bestimmen den Zustand der Klang-Warteschlangen.

s_AllThreshold	Alle Klang-Warteschlangen haben den Schwellenwert erreicht.

s_QueueEmpty Alle Klang-Warteschlangen sind leer und die Klangtreiber sind ausgeschaltet.

s_Threshold Eine Klang-Warteschlange hat den Schwellenwert erreicht und gibt den Klang aus.

sb_Xxxx

Diese Konstanten legen Bildlaufleistenereignisse fest und werden als Parameter in den Botschaften *wm_HScroll*, *wm_HScrollClipboard*, *wm_VScroll* und *wm_VScrollClipboard* übergeben.

sb_Bottom Bildlauf nach unten rechts.

sb_EndScroll Ende des Bildlaufs.

sb_LineDown Bildlauf eine Zeile nach unten.

sb_LineUp Bildlauf eine Zeile nach oben.

sb_PageDown Bildlauf eine Seite nach unten.

sb_PageUp Bildlauf eine Seite nach oben.

sb_ThumbPosition Bildlauf zu einer absoluten Position. Die aktuelle Position wird in *lParamLo* bereitgestellt.

sb_ThumbTrack Mauszug zur festgelegten Position. Die aktuelle Position wird in *lParamLo* bereitgestellt.

sb_Top Bildlauf nach oben links.

Diese Konstanten bezeichnen die Bildlaufleiste, die durch die Funktionen bzw. Prozeduren *GetScrollPos*, *GetScrollRange*, *SetScrollPos*, *SetScrollRange* und *ShowScrollBar* festgelegt werden.

sb_BOTH Bestimmt die horizontale und die vertikale Bildlaufleiste des Fensters.

sb_CTL Bestimmt eine selbständige Bildlaufleiste.

sb_HORZ Bestimmt die horizontale Bildlaufleiste des Fensters.

sb_VERT Bestimmt die vertikale Bildlaufleiste des Fensters.

sbs_Xxxx

Diese Konstanten werden verwendet, um den Bildlaufleistenstil festzulegen, wenn sie durch die Funktionen *CreateWindow* oder *CreateWindowEx* erzeugt werden.

sbs_BottomAlgin Wird mit dem Stil *sbs_Horz* verwendet. Der untere Rand der Bildlaufleiste wird am unteren Rand des Rechtecks ausgerichtet, das durch die in der Funktion *CreateWindow* angegebenen Parameter *X*, *Y*, *Width*

und *Height* festgelegt wird. Die Bildlaufleiste hat die voreingestellte Höhe für Systembildlaufleisten.

sbs_Horz Bestimmt eine horizontale Bildlaufleiste. Sind weder der Stil *sbs_BottomAlign* noch der Stil *sbs_TopAlign* gesetzt, hat die Bildlaufleiste die mit der Funktion *CreateWindow* gesetzte Höhe, Breite und Position.

sbs_LeftAlgin Wird mit dem Stil *sbs_Vert* benutzt. Der linke Rand der Bildlaufleiste wird nach dem linken Rand des Rechtecks ausgerichtet, das durch die mit der Funktion *CreateWindow* gesetzten Parameter *X*, *Y*, *Width* und *Height* festgelegt wird. Die Bildlaufleiste hat die voreingestellte Breite für Systembildlaufleisten.

sbs_RightAlgin Wird mit dem Stil *sbs_Vert* benutzt. Der rechte Rand der Bildlaufleiste wird nach dem rechten Rand des Rechtecks ausgerichtet, das durch die in der Funktion *CreateWindow* angegebenen Parameter *X*, *Y*, *Width* und *Height* festgelegt wird. Die Bildlaufleiste hat die voreingestellte Breite für Systembildlaufleisten.

sbs_SizeBox Bestimmt ein Größenveränderungsfeld. Wenn weder der Stil *sbs_SizeBoxBottomRightAlign* noch der Stil *sbs_SizeBoxTopLeftAlign* gesetzt sind, hat das Größenveränderungsfeld die in der Funktion *CreateWindow* gegebene Höhe, Breite und Position.

sbs_SizeBoxBottom-RightAlgin Wird mit dem Stil *sbs_SizeBox* verwendet. Die untere rechte Ecke des Größenveränderungsfeldes wird an der unteren rechten Ecke des Rechtecks ausgerichtet, das durch die in der Funktion *CreateWindow* angegebenen Parameter *X*, *Y*, *Width* und *Height* festgelegt wird. Das Größenveränderungsfeld hat die voreingestellte Größe des Systemgrößenveränderungsfeldes.

sbs_SizeBoxTopLeftAlgin Wird mit dem Stil *sbs_SizeBox* verwendet. Die obere linke Ecke des Größenveränderungsfeldes wird an der oberen linken Ecke des Rechtecks ausgerichtet, das durch die in der Funktion *CreateWindow* angegebenen Parameter *X*, *Y*, *Width* und *Height* festgelegt wird. Das Größenveränderungsfeld hat die voreingestellte Größe des Systemgrößenveränderungsfeldes.

sbs_TopAlgin Wird mit dem Stil *sbs_Horz* benutzt. Der obere Rand der Bildlaufleiste wird am oberen Rand des Rechtecks ausgerichtet, das durch die in der Funktion *CreateWindow* angegebenen Parameter *X*, *Y*, *Width* und *Height* festgelegt wird. Die Bildlaufleiste hat die voreingestellte Höhe für Systembildlaufleisten.

sbs_Vert	Bestimmt eine vertikale Bildlaufleiste. Ist weder der Stil *sbs_RightAlign* noch der Stil *sbs_LeftAlign* angegeben, dann hat die Bildlaufleiste die die in der Funktion *CreateWindow* angegebene Höhe, Breite und Position.

sc_Xxxx

Die folgenden Befehle werden in der *wm_SysCommand*-Botschaft als Reaktion auf eine Auswahl im Systemmenü oder auf einen Klick im Verkleinerungs- bzw. Vergrößerungsfeld übergeben. Sie zeigen die Anforderung an.

sc_Close	Fenster schließen.
sc_HScroll	Horizontalen Bildlauf durchführen.
sc_KeyMenu	Das dem Tastendruck entsprechende Menü darstellen.
sc_Maximize, sc_Zoom	Fenster als Vollbild darstellen.
sc_Minimize, sc_Icon	Fenster als Symbol darstellen.
sc_MouseMenu	Das dem Mausklick entsprechende Menü darstellen.
sc_Move	Das Fenster verschieben.
sc_NextWindow	In das nächste Fenster bewegen.
sc_PrevWindow	In das vorhergehende Fenster bewegen.
sc_Restore	Stellt nach einer Maximierung oder Minimierung die ursprüngliche Größe des Fensters wieder her.
sc_Size	Größe des Fensters einstellen.
sc_TaskList	Zeigt eine Task-Liste an.
sc_VScroll	Vertikalen Bildlauf durchführen.

Schnittstellenkonfigurations-Konstanten

Eine der folgenden Konstanten kann im Feld *Parity* eines *TDCB*-Record verwendet werden. Sie zeigen den Paritätstyp an, der von einer seriellen Schnittstelle verwendet werden soll.

Konstante	Parität
EvenParity	Gerade
MarkParity	Markierung
NoParity	Keine
OddParity	Ungerade
SpaceParity	Leer

Eine der folgenden Konstanten kann im Feld *StopBits* eines *TDCB*-Record verwendet werden. Sie zeigt die Anzahl von Stopbits an, die von einer seriellen Schnittstelle verwendet werden.

Konstante	Verwendete Stopbits
OneStopBit	1
One5StopBits	1,5
TwoStopBits	2

Schriftdarstellungsqualitäts-Flags

Diese Konstanten bestimmen die gewünschte Ausgabequalität für Schriften, die mit der Funktion *CreateFont* erzeugt werden.

Default_Quality Ausgabequalität ist unerheblich.

Draft_Quality Ausgabequalität ist von eingeschränktem Interesse.

Proof_Quality Die Ausgabequalität der Zeichen ist wichtiger als die Übereinstimmung mit virtuellen Schriftattributen.

ShowWindow-Befehle (frühere Versionen)

Dies sind ältere Konstanten, die nur aus Gründen der Abwärtskompatibilität mit früheren Windows-Versionen übernommen wurden. Wenn möglich, sollten die *sw_* Fensterdarstellungskonstanten verwendet werden.

Konstante	Wert	Bedeutung
hide_Window	0	Fenster verbergen.
show_OpenWindow	1	Ein Fenster aus seinem Symbol wiederherstellen.
show_IconWindow	2	Fenster zum Symbol verkleinern.
show_FullScreen	3	Fenster zum Vollbild vergrößern.
show_OpenNoActivate	4	Fenster wiederherstellen, aber nicht aktivieren.

SizeXxxx

Diese Konstanten zeigen die Art der angeforderten Fenstergrößenänderung an. Sie werden in *wm_Size*-Botschaften übergeben.

SizeFullScreen Fenster wurde zum Vollbild vergrößert.

SizeIconic Fenster wurde zum Symbol verkleinert.

SizeNormal Die Größe des Fensters wurde geändert, *SizeIconic* oder *SizeFullScreen* treffen jedoch nicht zu.

SizeZoomHide Die Botschaft wird an alle Pop-up-Fenster übergeben, wenn ein anderes Fenster zum Vollbild vergrößert wurde.

SizeZoomShow	Die Botschaft wird an alle Pop-up-Fenster übergeben, wenn ein anderes Fenster in seiner voreingestellten Größe wiederhergestellt wurde.

sm_Xxxx

Diese Codes bestimmen Eigenschaften der Windows-Anwenderschnittstelle. Die jeweiligen Abmessungen können über die Funktion *GetSystemMetrics* abgerufen werden.

sm_CXBorder	Breite des Fensterrahmens, nicht einstellbar.
sm_CXCursor	Breite des Zeigers.
sm_CXDlgFrame	Breite des Rahmens, wenn das Fenster den Stil *ws_DlgFrame* hat.
sm_CXFrame	Breite des Fensterrahmens, einstellbar.
sm_CXFullScreen	Breite eines Fenster-Client-Bereiches für eine Vollbildanwendung.
sm_CXHScroll	Breite des Pfeilbitmaps auf der horizontalen Bildlaufleiste.
sm_CXHThumb	Breite des Bildlaufsymbols auf der horizontalen Bildlaufleiste.
sm_CXIcon	Breite eines Symbols.
sm_CXMin	Minimale Breite des Fensters.
sm_CXMinTrack	Minimale Tracking-Breite des Fensters.
sm_CXScreen	Breite des Bildschirms.
sm_CXSize	Breite der Bitmaps, die in der Titelzeile enthalten sind.
sm_CXVScroll	Breite des Pfeilbitmaps auf der vertikalen Bildlaufleiste.
sm_CYBorder	Höhe des Fensterrahmens, nicht einstellbar.
sm_CYCaption	Höhe der Titelzeile eines Fensters.
sm_CYCursor	Höhe des Zeigers.
sm_CYDlgFrame	Höhe des Rahmens, wenn das Fenster den Stil *ws_DlgFrame* hat.
sm_CYFrame	Höhe des Fensterrahmens, einstellbar.
sm_CYFullScreen	Höhe des Fenster-Client-Bereiches für eine Vollbildanwendung (gleich der Höhe des Bildschirms minus der Höhe der Titelzeile).
sm_CYHScroll	Höhe des Pfeilbitmaps auf der horizontalen Bildlaufleiste.

sm_CYIcon	Höhe des Symbols.
sm_CYKanjiWindow	Höhe des Kanji-Fensters.
sm_CYMenu	Höhe einer einzeiligen Menüleiste.
sm_CYMin	Minimale Höhe des Fensters.
sm_CYMinTrack	Minimale Tracking-Höhe des Fensters.
sm_CYScreen	Höhe des Bildschirms.
sm_CYSize	Höhe der Bitmaps, die in der Titelzeile enthalten sind.
sm_CYVScroll	Höhe des Pfeilbitmaps auf der vertikalen Bildlaufleiste.
sm_CYVThumb	Höhe des Bildlaufsymbols auf der vertikalen Bildlaufleiste.
sm_Debug	Ungleich 0, wenn die Windows-Debugger-Version vorliegt.
sm_MousePresent	Ungleich 0, wenn eine Maus installiert ist.
sm_SwapButton	Ungleich 0, wenn die linke und rechte Maustaste vertauscht wurden.

sp_Xxxx

Diese Konstanten werden als Fehlercodes durch die *Escape*-Funktion zurückgegeben.

sp_AppAbort	Die Anwendung hat den Druckauftrag abgebrochen.
sp_Error	Allgemeiner Fehler.
sp_OutOfDisk	Speicherplatz auf der Diskette bzw. Platte nicht ausreichend für Spooling. Zusätzlicher Speicherplatz kann nicht bereitgestellt werden.
sp_OutOfMemory	Arbeitsspeicher ist für Spooling nicht ausreichend.
sp_UserAbort	Der Benutzer hat den Druckauftrag mit Hilfe des Druck-Managers abgebrochen.

ss_Xxxx

Diese Konstanten werden verwendet, um den Stil eines statischen Steuerelements festzulegen, wenn es durch die Funktionen *CreateWindow* oder *CreateWindowEx* erzeugt wird.

ss_BlackFrame	Bestimmt ein Feld mit einem Rahmen, der in der gleichen Farbe wie Fensterrahmen gezeichnet wird. Diese Farbe ist Schwarz im voreingestellten Windows-Farbschema.

ss_BlackRect Bestimmt ein Rechteck, das mit der gleichen Farbe gefüllt ist, mit der Fensterrahmen gezeichnet werden. Diese Farbe ist Schwarz im voreingestellten Windows-Farbschema.

ss_Center Bestimmt ein einfaches Rechteck und zeigt den gegebenen Text zentriert im Rechteck an. Der Text wird formatiert, bevor er angezeigt wird. Wörter, die über das Ende einer Zeile hinausreichen würden, werden automatisch auf den Anfang der nächsten zentrierten Zeile umbrochen.

ss_GrayFrame Bestimmt ein Feld mit einem Rahmen, der in der gleichen Farbe wie der Bildschirmhintergrund (Desktop) gezeichnet wird.

ss_GrayRect Bestimmt ein Rechteck, das mit der Farbe gefüllt wird, die benutzt wird, um den Bildschirmhintergrund zu zeichnen. Dies ist Grau im voreingestellten Windows-Farbschema.

ss_Icon Bestimmt ein Symbol, das im Dialogfenster angezeigt wird. Der gegebene Text ist der Name eines Symbols (kein Dateiname), der anderswo in der Ressourcendatei definiert ist. Die Parameter *Width* und *Height* werden ignoriert; das Symbol legt seine Größe selbst fest.

ss_Left Bestimmt ein einfaches Rechteck und zeigt den gegebenen Text linksbündig im Rechteck an. Der Text wird formatiert, bevor er angezeigt wird. Wörter, die über das Ende einer Zeile hinausreichen würden, werden automatisch auf den Anfang der nächsten Zeile umbrochen.

ss_LeftNoWordWrap Bestimmt ein einfaches Rechteck und zeigt den gegebenen Text linksbündig im Rechteck an. Tabulatorzeichen werden erweitert, aber Wörter werden nicht umbrochen. Text, der über das Ende einer Zeile hinausreicht, wird abgeschnitten.

ss_NoPrefix Wenn dieser Stil nicht angegeben wird, wird Windows jedes »&« Zeichen im Text des Steuerelements als Akzeleratorpräfixzeichen interpretieren. In diesem Fall wird das »&« entfernt und das nächste Zeichen im String unterstrichen. Soll ein statisches Steuerelement Text enthalten, bei dem dieses Merkmal unerwünscht ist, kann *ss_NoPrefix* hinzugefügt werden. Dieser statische Steuerelementstil kann in jedes der definierten Steuerelemente mit einbezogen werden. Man kann *ss_NoPrefix* mit anderen Stilen kombinie-

ren, indem man den bitweisen **or**-Operator benutzt. Dies wird meist genutzt, wenn Dateinamen oder andere Strings, die ein »&« enthalten können, in einem statischen Steuerelement in einem Dialogfenster angezeigt werden sollen.

ss_Right Bestimmt ein einfaches Rechteck und zeigt den gegebenen Text rechtsbündig im Rechteck an. Der Text wird formatiert, bevor er angezeigt wird. Wörter, die über das Ende einer Zeile hinausreichen würden, werden automatisch auf die nächste Zeile rechtsbündig umbrochen.

ss_Simple Bestimmt ein einfaches Rechteck und zeigt eine einzelne Textzeile linksbündig im Rechteck an. Die Textzeile kann auf keinen Fall gekürzt oder geändert werden. (Das übergeordnete Fenster oder das Dialogfenster des Steuerelements darf die Botschaft *wm_CtlColor* nicht bearbeiten.)

ss_UserItem Bestimmt ein anwenderdefiniertes, statisches Steuerelement.

ss_WhiteFrame Bestimmt ein Feld mit einem Rahmen, der in der gleichen Farbe wie der Fensterhintergrund gezeichnet wird. Diese Farbe ist Weiß im voreingestellten Windows-Farbschema.

ss_WhiteRect Bestimmt ein Rechteck, das mit der Farbe gefüllt wird, die benutzt wird, um den Fensterhintergrund zu zeichnen. Diese Farbe ist Weiß im voreingestellten Windows-Farbschema.

StretchBlt-Modi

Diese Konstanten sind die Bitmap-Streckmodi. Sie werden in den Funktionen *GetStretchBltMode* und *SetStretchBltMode* verwendet.

BlackOnWhite **and** in den getilgten Zeilen. Dieser Modus erhält schwarze auf Kosten der weißen Pixel durch den Gebrauch des Operators **and** mit den zu tilgenden und den verbleibenden Zeilen.

ColorOnColor Löscht die getilgten Zeilen. Dieser Modus löscht alle getilgten Zeilen und versucht dabei nicht, ihre Information zu erhalten.

WhiteOnBlack **or** in den getilgten Zeilen. Dieser Modus erhält weiße Pixel auf Kosten der schwarzen durch den Gebrauch des Operators **or** mit den zu tilgenden und den verbleibenden Zeilen.

sw_Xxxx

Diese Konstanten zeigen den Status an, mit dem die Funktion *ShowWindow* Fenster darstellt.

sw_Hide	Versteckt das Fenster und gibt die Aktivierung an ein anderes Fenster ab.
sw_Maximize	Wie *sw_ShowMaximized*.
sw_Minimize	Verkleinert das festgelegte Fenster und aktiviert das Hauptfenster in der Liste des Fenster-Managers.
sw_Normal	Wie *sw_ShowNormal*.
sw_OtherZoom	Ein anderes Fenster wird vergrößert. Wegen der Kompatibilität mit Windows 2.0 übernommen.
sw_OtherUnzoom	Ein anderes Fenster wird verkleinert. Wegen der Kompatibilität mit Windows 2.0 übernommen.
sw_Restore	Wie *sw_ShowNormal*.
sw_Show	Aktiviert ein Fenster und zeigt es in seiner aktuellen Größe und Position.
sw_ShowMaximized	Aktiviert das Fenster und zeigt es als vergrößertes Fenster.
sw_ShowMinimized	Aktiviert ein Fenster und zeigt es als Symbol.
sw_ShowMinNoActive	Zeigt das Fenster als Symbol. Das gegenwärtig aktive Fenster bleibt aktiv.
sw_ShowNA	Zeigt das Fenster in seinem aktuellen Zustand. Das gegenwärtig aktive Fenster bleibt aktiv.
sw_ShowNoActivate	Zeigt ein Fenster in seiner neuesten Größe und Position. Das gegenwärtig aktive Fenster bleibt aktiv.
sw_ShowNormal	Aktiviert ein Fenster und zeigt es an. Ist das Fenster verkleinert oder vergrößert, so stellt Windows es in seiner ursprünglichen Größe und Position wieder her.

Diese Bezeichner geben den Status des darzustellenden Fensters an und werden in der *wm_ShowWindow*-Botschaft übergeben.

sw_ParentClosing	Es wird ein übergeordnetes Fenster geschlossen (Symbol) oder ein Pop-up-Fenster verborgen.
sw_ParentOpening	Es wird ein übergeordnetes Fenster geöffnet oder ein Pop-up-Fenster gezeigt.

swp_Xxxx

Diese Flags werden in der Prozedur *SetWindowPos* und der Funktion *DeferWindowPos* übergeben, um besondere Aktionen für das angegebene Fenster zu vereinbaren.

swp_DrawFrame	Zeichnet den in der Fensterklassendefinition definierten Rahmen um das Fenster.
swp_HideWindow	Versteckt das Fenster.
swp_NoActivate	Fenster wird nicht aktiviert.
swp_NoMove	Erhält die aktuelle Position, ignoriert die Parameter *x* und *y*.
swp_NoRedraw	Änderungen werden nicht dargestellt.
swp_NoSize	Erhält die aktuelle Größe, ignoriert die Parameter *cx* und *cy*.
swp_NoZOrder	Erhält die aktuelle Ordnung, ignoriert den Parameter *WndInsertAfter*.
swp_ShowWindow	Stellt das Fenster dar.

syspal_Xxxx

Diese Konstanten bestimmen die neue Verwendung der Sytempalette. Sie werden in den Funktionen *GetSystemPaletteUse* und *SetSystemPaletteUse* übergeben.

syspal_NoStatic	Die Systempalette enthält keine statischen Farben außer Schwarz und Weiß.
syspal_Static	Die Systempalette enthält statische Farben, die sich nicht ändern, wenn eine Anwendung ihre virtuelle Palette realisiert.

ta_Xxxx

Diese Ausrichtungsoptionen steuern die Ausrichtung von Text, der mit einer der Funktionen *TextOut* oder *ExtTextOut* dargestellt wird. Die Optionen werden in einer der Funktionen *GetTextAlign* oder *SetTextAlign* festgelegt oder zurückgegeben.

ta_BaseLine	Bestimmt die Ausrichtung der x-Achse an der Grundlinie der ausgewählten Schrift im umgebenden Rechteck.
ta_Bottom	Bestimmt die Ausrichtung der x-Achse an der Unterseite des umgebenden Rechtecks.

ta_Center	Bestimmt die Ausrichtung der y-Achse am Mittelpunkt des umgebenden Rechtecks.
ta_Left	Bestimmt die Ausrichtung der y-Achse an der linken Seite des umgebenden Rechtecks.
ta_NoUpdateCP	Bestimmt, daß die aktuelle Position nicht nach jedem Aufruf der Funktionen *TextOut* oder *ExtTextOut* aktualisiert wird.
ta_Right	Bestimmt die Ausrichtung der y-Achse an der rechten Seite des umgebenden Rechtecks.
ta_Top	Bestimmt die Ausrichtung der x-Achse an der Oberseite des umgebenden Rechtecks.
ta_UpdateCP	Bestimmt, daß die aktuelle Position nach jedem Aufruf der Funktionen *TextOut* oder *ExtTextOut* aktualisiert wird.

tc_Xxxx

Diese Konstanten repräsentieren die Textdarstellungsfähigkeiten des Geräts.

tc_cp_Stroke	Gerät kann vektoriell mit Clipping ausgeben.
tc_cr_90	Gerät kann Buchstaben um 90 Grad rotieren.
tc_cr_Any	Gerät kann beliebige Buchstabenrotationen vornehmen.
tc_ea_Double	Gerät kann Zeichen fett darstellen.
tc_ia_Able	Gerät kann kursive Zeichen darstellen.
tc_op_Character	Gerät kann Zeichen genau ausgeben.
tc_op_Stroke	Gerät kann Zeichen vektoriell ausgeben.
tc_ra_Able	Gerät kann Rasterschriften darstellen.
tc_sa_Contin	Gerät kann einen beliebigen vielfachen Wert zur Skalierung verwenden.
tc_sa_Double	Gerät kann mit zweifachem Zeichenabstand skalieren.
tc_sa_Integer	Gerät kann das Mehrfache eines Integerwerts zur Skalierung verwenden.
tc_sf_X_YIndep	Gerät kann unabhängig von X und Y skalieren.
tc_so_Able	Gerät kann Zeichen durchgestrichen darstellen.
tc_ua_Able	Gerät kann unterstreichen.
tc_va_Able	Gerät kann Vektorschriften darstellen.

Ternäre Rasteroperationen

Die folgenden Konstanten werden als Rasteroperationscodes in den Funktionen *BitBlt, PatBlt, StretchBlt* und *StretchDIBits* verwendet.

Blackness	Alle Ausgaben werden schwarz dargestellt.
DstInvert	Invertiert das Ziel-Bitmap.
MergeCopy	Kombiniert Vorgabe- und Quell-Bitmap unter Verwendung des logischen **and**-Operators.
MergePaint	Kombiniert das invertierte Quell-Bitmap mit dem Ziel-Bitmap unter Verwendung des logischen **or**-Operators.
NotSrcCopy	Kopiert das invertierte Ziel-Bitmap ins Ziel.
NotSrcErase	Invertiert das Resultat der Kombination von Quell- und Ziel-Bitmap unter Verwendung des logischen **or**-Operators.
PatCopy	Kopiert die Vorgabe in das Ziel-Bitmap.
PatInvert	Kombiniert das Ziel-Bitmap mit der Vorgabe unter Verwendung des logischen **xor**-Operators.
PatPaint	Kombiniert das invertierte Quell-Bitmap mit der Vorgabe unter Verwendung des logischen **or**-Operators. Das Resultat wird mit dem Ziel-Bitmap unter Verwendung des logischen **or**-Operators kombiniert.
SrcAnd	Kombiniert Pixel des Ziel- und Quell-Bitmaps unter Verwendung des logischen **and**-Operators.
SrcCopy	Kopiert das Quell- in das Ziel-Bitmap.
SrcErase	Invertiert das Ziel-Bitmap und kombiniert das Resultat mit dem Quell-Bitmap unter Verwendung des logischen **and**-Operators.
SrcInvert	Kombiniert Pixel von Ziel- und Quell-Bitmap unter Verwendung des logischen **xor**-Operators.
SrcPaint	Kombiniert Pixel von Ziel- und Quell-Bitmap unter Verwendung des logischen **or**-Operators.
Whiteness	Färbt jede Ausgabe weiß.

tf_ForceDrive

Wird dieses Flag mit dem Argument *DriveLetter* in einem Aufruf der Funktion *GetTempFileName* kombiniert, stellt es sicher, daß die temporäre Datei auf dem angegebenen Laufwerk erzeugt wird. Andernfalls wird diese Datei auf der ersten Festplatte oder im Verzeichnis der Environment-Variablen TEMP erzeugt.

vk_Xxxx

Virtuelle Tastencodes sind Konstanten, die zur Repräsentierung der Standardtasten (z.B. »A« oder »F1«) des Computers verwendet werden. Da verschiedene Computermodelle unterschiedliche Tasten haben können, werden die virtuellen Tastencodes verwendet, um die Lauffähigkeit einer Anwendung auf verschiedenen Computern zu ermöglichen. Diese Codes werden zur Definition von Akzeleratoren oder zur Bearbeitung der Tastaturbotschaften *wm_KeyDown*, *wm_KeyUp*, *wm_SysKeyDown* und *wm_SysKeyUp* verwendet.

Die folgende Tabelle zeigt die virtuellen Tastencodes und die ihnen entsprechenden Tasten.

Bezeichnung	Wert	Beschreibung
vk_LButton	01H	Linke Maustaste
vk_RButton	02H	Rechte Maustaste
vk_Cancel	03H	Für die Verarbeitung von Control-Break
vk_MButton	04H	Mittlere Maustaste (Dreitastenmaus)
	05H-07H	Nicht definiert
vk_Back	08H	Backspace -Taste
vk_Tab	09H	Tab -Taste
	0AH-0BH	Nicht definiert
vk_Clear	0CH	Löschtaste (Bildschirm)
vk_Return	0DH	Return -Taste
vk_Shift	10H	Shift -Taste
vk_Control	11H	Control -Taste
vk_Menu	12H	Menütaste
vk_Pause	13H	Pause -Taste
vk_Capital	14H	CapsLock -Taste
	15H-19H	Reserviert für Kanji-Systeme
	1AH	Nicht definiert
vk_Escape	1BH	Escape -Taste
	CH-1FH	Rerserviert für Kanji-Systeme
vk_Space	20H	Leertaste
vk_Prior	21H	PgUp -Taste
vk_Next	22H	PgDn -Taste
vk_End	23H	End -Taste
vk_Home	24H	Home -Taste
vk_Left	25H	Cursortaste Links
vk_Up	26H	Cursortaste Oben
vk_Right	27H	Cursortaste Rechts
vk_Down	28H	Cursortaste Unten
vk_Select	29H	Auswahltaste (Select)
vk_Print	2AH	OEM-spezifisch
vk_Execute	2BH	Taste Ausführen (Execute)
vk_SnapShot	2CH	PrintScreen für Windows Version 3.0 und höher
vk_Insert	2DH	Ins -Taste

Bezeichnung	Wert	Beschreibung
vk_Delete	2EH	Del -Taste
vk_Help	2FH	Hilfetaste
vk_0	30H	Taste 0
vk_1	31H	Taste 1
vk_2	32H	Taste 2
vk_3	33H	Taste 3
vk_4	34H	Taste 4
vk_5	35H	Taste 5
vk_6	36H	Taste 6
vk_7	37H	Taste 7
vk_8	38H	Taste 8
vk_9	39H	Taste 9
	3AH-40H	Nicht definiert
vk_A	41H	Taste A
vk_B	42H	Taste B
vk_C	43H	Taste C
vk_D	44H	Taste D
vk_E	45H	Taste E
vk_F	46H	Taste F
vk_G	47H	Taste G
vk_H	48H	Taste H
vk_I	49H	Taste I
vk_J	4AH	Taste J
vk_K	4BH	Taste K
vk_L	4CH	Taste L
vk_M	4DH	Taste M
vk_N	4EH	Taste N
vk_O	4FH	Taste O
vk_P	50H	Taste P
vk_Q	51H	Taste Q
vk_R	52H	Taste R
vk_S	53H	Taste S
vk_T	54H	Taste T
vk_U	55H	Taste U
vk_V	56H	Taste V
vk_W	57H	Taste W
vk_X	58H	Taste X
vk_Y	59H	Taste Y
vk_Z	5AH	Taste Z
	5BH-5FH	Nicht definiert
vk_Numpad0	60H	Numerisches Tastenfeld, Taste 0
vk_Numpad1	61H	Numerisches Tastenfeld, Taste 1
vk_Numpad2	62H	Numerisches Tastenfeld, Taste 2
vk_Numpad3	63H	Numerisches Tastenfeld, Taste 3
vk_Numpad4	64H	Numerisches Tastenfeld, Taste 4

Bezeichnung	Wert	Beschreibung
vk_Numpad5	65H	Numerisches Tastenfeld, Taste 5
vk_Numpad6	66H	Numerisches Tastenfeld, Taste 6
vk_Numpad7	67H	Numerisches Tastenfeld, Taste 7
vk_Numpad8	68H	Numerisches Tastenfeld, Taste 8
vk_Numpad9	69H	Numerisches Tastenfeld, Taste 9
vk_Multiply	6AH	Multiplikationstaste
vk_Add	6BH	Additionstaste
vk_Separater	6CH	Trenntaste
vk_Subtract	6DH	Subtraktionstaste
vk_Decimal	6EH	Dezimaltrennzeichentaste
vk_Divide	6FH	Divisionstaste
vk_F1	70H	F1
vk_F2	71H	F2
vk_F3	72H	F3
vk_F4	73H	F4
vk_F5	74H	F5
vk_F6	75H	F6
vk_F7	76H	F7
vk_F8	77H	F8
vk_F9	78H	F9
vk_F10	79H	F10
vk_F11	7AH	F11
vk_F12	7BH	F12
vk_F13	7CH	F13
vk_F14	7DH	F14
vk_F15	7EH	F15
vk_F16	7FH	F16
	80H-87H	OEM-spezifisch
	88H-8FH	Nicht belegt
vk_Numlock	90H	Taste NumLock
vk_OEM_Scroll	91H	Taste ScrollLock
	92H-B9H	Nicht belegt
vk_OEM_1	BAH	Tastaturspezifische Interpunktionstaste (nicht auf jeder Tastatur vorhanden)
vk_OEM_Plus	BBH	Taste Plus +
vk_OEM_Comma	BCH	Taste Komma ,
vk_OEM_Minus	BDH	Taste Minus -
vk_OEM_Period	BEH	Taste Punkt .
vk_OEM_2	BFH	Tastaturspezifische Interpunktionstaste (nicht auf jeder Tastatur vorhanden)
vk_OEM_3	C0H	Tastaturspezifische Interpunktionstaste (nicht auf jeder Tastatur vorhanden)
	C1H-DAH	Nicht belegt
vk_OEM_4	DBH	Tastaturspezifische Interpunktionstaste (nicht auf jeder Tastatur vorhanden)

Bezeichnung	Wert	Beschreibung
vk_OEM_5	DCH	Tastaturspezifische Interpunktionstaste (nicht auf jeder Tastatur vorhanden)
vk_OEM_6	DDH	Tastaturspezifische Interpunktionstaste (nicht auf jeder Tastatur vorhanden)
vk_OEM_7	DEH	Tastaturspezifische Interpunktionstaste (nicht auf jeder Tastatur vorhanden)
vk_OEM_8	DFH	Tastaturspezifische Interpunktionstaste (nicht auf jeder Tastatur vorhanden)
	E0H-E1H	OEM-spezifisch
vk_OEM_102	E2H	<> oder | auf erweiterter, nicht US-IBM-kompatibler Tastatur mit 102-Tasten
	E3H-E4H	OEM-spezifisch
	E5H	Nicht belegt
	E6H	OEM-spezifisch
	E7H-E8H	Nicht belegt
	E9H-F5H	OEM-spezifisch
	F6H-FEH	Nicht belegt

Vordefinierte Zeichenobjekte

Diese Konstanten repräsentieren vordefinierte (Stock) Zeichenobjekte. Sie werden in der Funktion *GetStockObject* verwendet.

Black_Brush	Schwarzer Pinsel.
DkGray_Brush	Dunkelgrauer Pinsel.
Gray_Brush	Grauer Pinsel.
Hollow_Brush	Neutraler Pinsel.
LtGray_Brush	Hellgrauer Pinsel.
Null_Brush	Neutraler Pinsel.
White_Brush	Weißer Pinsel.
Black_Pen	Schwarzer Stift.
Null_Pen	Null-Stift.
White_Pen	Weißer Stift.
ANSI_Fixed_Font	ANSI-festgelegte Systemschrift.
ANSI_Var_Font	ANSI-variable Systemschrift.
Devide_Default_Font	Geräteabhängige Schrift.
OEM_Fixed_Font	OEM-abhängige festgelegte Schrift.
System_Fixed_Font	Die Systemschrift mit festem Zeichenabstand, die in

früheren Windows-Versionen benutzt wurde. Nur aus Gründen der Kompatibilität verfügbar.

System_Font Die Systemschrift. Standardmäßig benutzt Windows die Systemschrift, um Menüs, Dialogfenster und anderen Text zu zeichnen. In den Windows-Versionen 3.0 und höher ist die Systemschrift eine Proportionalschrift; frühere Windows-Versionen benutzen eine Systemschrift mit festem Zeichenabstand.

Default_Palette Standardfarbpalette. Diese Farbpalette besteht aus den 20 statischen Farben, die immer in der Systempalette präsent sind, um Farben in der virtuellen Palette von Hintergrundfenstern anzugleichen.

wep_Xxxx

Diese Codes werden in der Variablen *ExitCode* verwendet, auf die aus einer dynamischen Linkbibliothek zugegriffen werden kann.

wep_Free_DLL Zeigt an, daß nur die dynamische Linkbibliothek beendet und aus dem Speicher entfernt wird.

wep_System_Exit Zeigt an, daß Windows beendet wird.

wf_Xxxx

Diese Konstanten bestimmen die aktuelle Speicherkonfiguration von Windows. Sie werden als Rückgabewerte der Funktion *GetWinFlags* verwendet.

wf_80x87 System enthält einen mathematischen Intel-Coprozessor.

wf_CPU086 System-CPU ist ein 8086.

wf_CPU186 System-CPU ist ein 80186.

wf_CPU286 System-CPU ist ein 80286.

wf_CPU386 System-CPU ist ein 80386.

wf_CPU486 System-CPU ist ein 80486.

wf_Enhanced Windows läuft im erweiterten 386-Modus. Das Flag *wf_PMode* ist immer gesetzt, wenn *wf_Enhanced* gesetzt ist.

wf_LargeFrame Windows läuft mit Large-Frame-EMS-Speicherkonfiguration.

wf_Pmode Windows läuft im Protected Mode. Dieses Flag ist immer gesetzt, wenn entweder *wf_Enhanced* oder *wf_Standard* gesetzt ist.

wf_SmallFrame	Windows läuft mit Small-Frame-EMS-Speicherkonfiguration.
wf_Standard	Windows läuft im Standard Modus. *wf_Pmode* ist immer gesetzt, wenn *wf_Standard* gesetzt ist.
wf_Win286	Wie *wf_Standard*.
wf_Win386	Wie *wf_Enhanced*.

wh_Xxxx

Diese Codes legen einen bestimmten Typ der zu installierenden Filterfunktion fest (*SetWindowsHook*) oder entfernen ihn (*UnhookWindowsHook*) aus der Kette der Filterfunktionen.

wh_CallWndProc	Installiert einen Fensterfunktionsfilter.
wh_GetMessage	Installiert einen Botschaftsfilter.
wh_JournalPlayback	Installiert einen Wiedergabeprotokollfilter.
wh_JournalRecord	Installiert einen Aufzeichnungsprotokollfilter.
wh_Keyboard	Installiert einen Tastaturfilter.
wh_MsgFilter	Installiert einen Botschaftsfilter (nur *SetWindowsHook*).
wh_SysMsgFilter	Installiert einen systemweiten Botschaftsfilter (nur *SetWindowsHook*).

ws_Xxxx

Diese Konstanten werden in verschiedenen Kombinationen verwendet, um Fensterstile festzulegen, wenn Fenster, Dialogfenster oder Steuerelemente mit der Funktion *CreateWindow* oder *CreateWindowEx* erzeugt werden.

ws_Border	Erzeugt ein Fenster, das eine Umrandung hat.
ws_Caption	Erzeugt ein Fenster, das eine Titelleiste hat (hat den Stil *ws_Border* zur Folge). Dieser Stil kann nicht mit dem Stil *ws_DlgFrame* verwendet werden.
ws_Child	Erzeugt ein untergeordnetes Fenster. Kann nicht mit dem Stil *ws_Popup* verwendet werden.
ws_ChildWindow	Erzeugt ein untergeordnetes Fenster, das den Stil *ws_Child* hat.
ws_ClipChildren	Schließt den von untergeordneten Fenstern belegten Bereich aus, wenn innerhalb des übergeordneten Fensters gezeichnet wird. Wird benutzt, wenn das übergeordnete Fenster erzeugt wird.

ws_ClipSiblings Schneidet untergeordnete Fenster relativ zueinander ab; das heißt, wenn ein bestimmtes untergeordnetes Fenster eine *wmPaint*-Botschaft erhält, schneidet der Stil *ws_ClipSiblings* alle anderen überlappten untergeordneten Fenster aus der Region des untergeordneten Fensters heraus, das aktualisiert werden soll.
Wenn *ws_ClipSiblings* nicht gesetzt ist und untergeordnete Fenster überlappen sich, so ist es beim Zeichnen innerhalb des Client-Bereichs eines untergeordneten Fensters möglich, auch innerhalb des Client-Bereichs eines benachbarten untergeordneten Fensters zu zeichnen.) Nur zum Gebrauch mit dem Stil *ws_Child*.

ws_Disabled Erzeugt ein Fenster, das anfänglich nicht aktiviert ist.

ws_DlgFrame Erzeugt ein Fenster mit einer doppelten Umrandung, aber ohne Titelleiste.

ws_Group Bestimmt das erste Steuerelement aus einer Gruppe von Elementen, in der der Benutzer sich mit den Richtungstasten von einem Steuerelement zum nächsten bewegen kann.
Alle Steuerelemente, die nach dem ersten Element mit dem Stil *ws_Group* definiert wurden, gehören zur gleichen Gruppe. Das nächste Steuerelement mit dem Stil *ws_Group* beendet die Stilgruppe und beginnt die nächste Gruppe (das heißt, eine Gruppe hört da auf, wo die nächste beginnt). Nur Dialogfenster verwenden diesen Stil.

ws_HScroll Erzeugt ein Fenster, das eine horizontale Bildlaufleiste hat.

ws_Iconic Erzeugt ein Fenster, das anfangs als Symbol dargestellt wird. Nur zum Gebrauch mit dem Stil *ws_Overlapped*.

ws_Maximize Erzeugt ein Fenster von maximaler Größe.

ws_MaximizeBox Erzeugt ein Fenster, das ein Feld zur Ausdehnung auf größtmöglichen Umfang hat.

ws_Minimize Erzeugt ein Fenster von minimaler Größe.

ws_MinimizeBox Erzeugt ein Fenster, das ein Feld zur Reduzierung auf minimale Größe hat.

ws_Overlapped Erzeugt ein überlapptes Fenster. Ein überlapptes Fenster hat eine Überschrift und eine Umrandung.

ws_OverlappedWindow Erzeugt ein überlapptes Fenster, das die Stile *ws_Overlapped*, *ws_Caption*, *ws_SysMenu*, *ws_ThickFrame*, *ws_MinimizeBox* und *ws_MaximizeBox* hat.

ws_Popup	Erzeugt ein Pop-up-Fenster. Kann nicht mit dem Stil *ws_Child* benutzt werden.
ws_PopupWindow	Erzeugt ein Pop-up-Fenster, das die Stile *ws_Border*, *ws_Popup* und *ws_SysMenu* hat. Der Stil *ws_Caption* muß mit dem Stil *ws_PopupWindow* kombiniert werden, um das Systemmenü sichtbar zu machen.
ws_SizeBox	Wie *ws_ThickFrame*.
ws_SysMenu	Erzeugt ein Fenster, das ein Systemmenü in seiner Titelleiste hat. Wird nur für Fenster mit Titelleisten benutzt.
ws_TabStop	Gibt eines der Steuerelemente an, durch die sich der Benutzer mit der Tab -Taste bewegen kann. Die Tab -Taste bewegt den Fokus zum nächsten Steuerelement, das durch den Stil *ws_TabStop* angegeben wird. Nur Dialogfenster verwenden diesen Stil.
ws_ThickFrame	Erzeugt ein Fenster mit einem dicken Rahmen, der benutzt werden kann, um die Größe des Fensters zu verändern.
ws_Tiled	Wie *ws_Overlapped*.
ws_TiledWindow	Wie *ws_OverlappedWindow*.
ws_Visible	Erzeugt ein Fenster, das anfänglich sichtbar ist. Dies wird auf überlappte und Pop-up-Fenster angewendet. Bei überlappten Fenstern wird der Parameter *Y* für die Funktion *ShowWindow* als Parameter verwendet.
ws_VScroll	Erzeugt ein Fenster, das eine vertikale Bildlaufleiste hat.

ws_ex_Xxxx

Diese Konstanten werden im Parameter *exStyle* verwendet, um den erweiterten Stil eines Fensters festzulegen, wenn ein Fenster mit der Funktion *CreateWindowEx* erzeugt wird.

ws_ex_DlgModalFrame	Bestimmt ein Fenster mit einer doppelten Umrandung, das wahlweise mit einer Titelleiste erzeugt werden kann, indem man den Stil *ws_Caption* im Parameter *Style* angibt.
ws_ex_NoParentNotify	Bestimmt, daß ein untergeordnetes Fenster, das mit diesem Stil erzeugt wird, keine Botschaft *wm_ParentNotify* an sein übergeordnetes Fenster sendet, wenn das untergeordnete Fenster erzeugt oder zerstört wird.

Zeichenabstand-Flags

Diese Konstanten bestimmen den Zeichenabstand (proportional oder fest) für Schriften, die mit der Funktion *CreateFont* erzeugt werden.

Default_Pitch
Fixed_Pitch
Variable_Pitch

Zeichensatz-Flags

Diese Konstanten bestimmen den gewünschten Zeichensatz für Schriften, die mit der Funktion *CreateFont* erzeugt werden.

ANSI_CharSet
Symbol_CharSet
ShiftJIS_CharSet
OEM_CharSet

2. Windows-Funktionen

Dieses Kapitel enthält eine alphabetische Auflistung aller Funktionen und Prozeduren von Windows mit speziellen Informationen über die Windows-API.

Für jede Prozedur oder Funktion werden die im folgenden Beispiel exemplarisch dargestellten Informationen aufgeführt. Im einzelnen sind dies: der Kopf der Prozedur oder Funktion, wie er in der Unit *WinProcs* deklariert ist, und eine Beschreibung der ausgeführten Operation, der dabei verwendeten Parameter und des Rückgabewerts.

Beispiel

Syntax	`function Beispiel(AParam: AType; BParam: PBType): Word;`
Beschreibung	Eine kurze Beschreibung der Verwendung der Funktion *Beispiel*. *AParam:* Beschreibung des ersten Parameters. *BParam:* Beschreibung des zweiten Parameters.
Rückgabewert	Gibt den Word-Wert an, der von *Beispiel* zurückgeliefert wird, abhängig davon, ob die Funktion erfolgreich ausgeführt werden konnte oder nicht.
Hinweis	Enthält Informationen darüber, was beim Aufruf dieser Funktion bzw. Prozedur zu beachten ist.
Callback-Funktion	Gegebenenfalls eine Beschreibung der Callback-Funktion, die von der Funktion *Beispiel* benötigt wird.
Querverweis	Ein Hinweis auf verwandte Funktionen oder Prozeduren.

AccessResource

Syntax	`function AccessResource(Instance, ResInfo: THandle): Integer;`
Beschreibung	Diese Funktion öffnet die angegebene Ressourcendatei und stellt den Dateizeiger auf die gewünschte Ressource, um der Anwendung das Lesen dieser Ressource zu ermöglichen.

AccessResource unterstützt ein DOS-Datei-Handle, das in nachfolgenden Dateiaufrufen zum Laden der Ressource verwendet werden kann. Die Datei wird nur zum Lesen geöffnet.

Anwendungen, die Ressourcen mit Hilfe von *AccessResource* einlesen, müssen die Ressourcendatei anschließend durch Aufruf der Funktion *_lclose* wieder schließen.

Instance: Bezeichnet die Instanz des Moduls, in dessen ausführbarer Datei die Ressource enthalten ist.

ResInfo: Bezeichnet die gewünschte Ressource. Dieses Handle sollte mit Hilfe der Funktion *FindResource* erzeugt werden.

Rückgabewert Bei erfolgreicher Ausführung wird das DOS-Datei-Handle zur vorgegebenen Ressourcendatei, zurückgeliefert. Der Rückgabewert ist -1, wenn die Ressource nicht gefunden werden kann.

Hinweis *AccessResource* braucht die verfügbaren DOS-Datei-Handles möglicherweise völlig auf und verursacht Fehler, wenn die geöffnete Datei nach dem Zugriff auf die Ressource nicht wieder geschlossen wird.

Querverweis *FindResource*

AddAtom

Syntax
```
function AddAtom(Str: PChar): TAtom;
```

Beschreibung Diese Funktion fügt den durch den Parameter *Str* bezeichneten nullterminierte String in die Atom-Tabelle ein. Gleichzeitig wird ein neues Atom erzeugt, das den String bezeichnet. Das Atom kann in nachfolgenden Aufrufen der Funktion *GetAtomName* dazu verwendet werden, um den String aus der Atom-Tabelle abzurufen.

AddAtom speichert nur eine einzige Kopie des vorgegebenen Strings in der Atom-Tabelle. Existiert der String bereits in der Tabelle, gibt die Funktion den zugehörigen Atom-Wert zurück und erhöht den Referenzzähler des Strings um 1.

Str: Zeigt auf den in die Tabelle einzufügenden nullterminierten String.

Rückgabewert Bei erfolgreicher Ausführung der Funktion wird das erzeugte Atom zurückgegeben, andernfalls ist der Rückgabewert 0.

Hinweis Die von *AddAtom* zurückgegebenen Atom-Werte bewegen sich zwischen $C000 und $FFFF. Atome ignorieren Klein- und Großschreibung.

Querverweis *GetAtomName*

AddFontResource

Syntax
```
function AddFontResource(FileName: PChar): Integer;
```

Beschreibung Diese Funktion fügt die Schriftressource aus der durch den Parameter *FileName* bezeichneten Datei in die Schriftentabelle von Windows ein. Die Schrift kann anschließend von jeder Anwendung verwendet werden.

FileName: Zeigt auf einen String, der den Namen der Schriftressourcendatei oder ein Handle zum geladenen Modul enthält. Zeigt *FileName* auf den Namen der Schriftressourcendatei, muß es sich um einen nullterminierten String im DOS-Dateinamensformat handeln. Die Namenserweiterung muß ebenfalls angegeben sein. Enthält *FileName* ein Handle, so liegt es im niederwertigen Word, das höherwertige Word ist 0.

Rückgabewert Es wird die Anzahl der hinzugefügten Schriften zurückgeliefert. Wurden keine Schriften geladen, ist der Rückgabewert 0.

Hinweis Jede Anwendung, die Schriften in die Schriftentabelle von Windows einfügt oder aus ihr entfernt, sollte diese Änderung anderen Fenstern mitteilen. Hierzu wird die Funktion *SendMessage* mit auf -1 gesetztem Paramater *Wnd* verwendet, die die Botschaft *wm_FontChange* an alle Hauptfenster übergibt.

Anwendungen sollte alle durch sie hinzugefügten Schriftressourcen entfernen, wenn diese nicht mehr benötigt werden.

Querverweis *wm_FontChange*

AdjustWindowRect

Syntax
```
procedure AdjustWindowRect(var Rect: TRect; Style:
LongInt; Menu: Bool);
```

Beschreibung Diese Prozedur berechnet die Größe des Fensters auf Grundlage der gewünschten Größe des Client-Bereichs. Danach kann das Fensterrechteck an die Funktion *CreateWindow* übergeben werden, die ein Fenster mit einem Client-Bereich der gewünschten Größe erzeugt.

Ein Client-Rechteck ist das kleinstmögliche den Client-Bereich umschließende Rechteck. Ein Fensterrechteck ist das kleinstmögliche das Fenster vollständig umschließende Rechteck. Die Größe des berechneten Fensterrechtecks hängt vom Fensterstil ab sowie davon, ob das Fenster ein Menü hat.

Rect: TRect-Datenstruktur, die die Koordinaten des Client-Rechtecks enthält.

Style: Bestimmt den Stil des Fensters, dessen Client-Rechteck konvertiert werden soll.

Menu: True, wenn das Fenster ein Menü hat.

Hinweis Diese Prozedur geht von einer einzelnen Menüzeile aus. Wenn sich das Menü über mehrere Zeilen erstreckt , sind die Koordinaten nicht korrekt.

Querverweis *CreateWindow*

AdjustWindowRectEx

Syntax
```
procedure AdjustWindowRectEx(var Rect: TRect; Style:
LongInt; Menu: Bool; ExtStyle: LongInt);
```

Beschreibung Diese Prozedur berechnet die benötigte Größe eines Fensterrechtecks des erweiterten Stils auf Grundlage der gewünschten Größe des Client-Rechtecks. Danach kann das Fensterrechteck an die Funktion *CreateWindowEx* übergeben werden, die ein Fenster mit einem Client-Bereich der gewünschten Größe erzeugt.

Ein Client-Rechteck ist das kleinstmögliche den Client-Bereich umschließende Rechteck. Ein Fensterrechteck ist das kleinstmögliche das Fenster vollständig umschließende Rechteck. Die Größe des berechneten Fensterrechtecks hängt vom Fensterstil ab sowie davon, ob das Fenster ein Menü hat.

Rect: TRect-Datenstruktur, die die Koordinaten des Client-Rechtecks enthält.

Style: Bestimmt den Stil des Fensters, dessen Client-Rechteck konvertiert werden soll.

Menu: True, wenn das Fenster ein Menü hat.

ExtStyle: Bestimmt den erweiterten Stil des zu erzeugenden Fensters.

Hinweis Diese Prozedur geht von einer einzelnen Menüzeile aus. Wenn sich das Menü über mehrere Zeilen erstreckt, sind die Koordinaten nicht korrekt.

Querverweis *CreateWindowEx*

AllocDStoCSAlias

Syntax
```
function AllocDStoCSAlias(Selector: Word) Word;
```

Beschreibung Diese Funktion übernimmt den Selektor eines Datensegments und gibt den Selektor eines Codesegments zurück. Wird im Protected Mode gearbeitet, führt der Versuch, Code in einem Datensegment auszuführen, zu einem schweren Fehler. *AllocDStoCSAlias* erlaubt die Ausführung von Code, der von der Anwendung im eigenen Stack-Segment erzeugt wurde.

Die Anwendung muß den neuen Selektor durch Aufruf der Funktion *FreeSelector* freigeben.

Selector: Bestimmt den Selektor des Datensegments

Rückgabewert Der Rückgabewert ist der dem Selektor des Datensegments entsprechende Selektor des Codesegments. Kann die Funktion keinen neuen Selektor zuweisen, ist der Rückgabewert 0.

Hinweis Windows überwacht Segmentverschiebungen nicht, aus diesem Grund muß das Datensegment fest sein und darf keinesfalls verschoben werden. Eine Verschiebung des Datensegments könnte fehlerhafte Veränderungen des Codesegments nach sich ziehen.

Mit der Funktion *ChangeSelector* kann der einem Daten-Selektor entsprechenden Code-Selektor auch ermittelt werden.

Anwendungen sollten *AllocDStoCSAlias* nur in Ausnahmefällen verwenden. Die Verwendung dieser Funktion verletzt alle empfohlenen Praktiken der Windows-Programmierung.

AllocResource

Syntax
```
function AllocResource(Instance, THandle; ResInfo:
Size: LongInt): THandle;
```

Beschreibung Diese Funktion reserviert nichtinitialisierten Speicher für die übergebene Ressource, diese Reservierung muß für alle Ressourcen vorgenommen werden. *AllocResource* wird vor dem Laden einer Ressource durch die Funktion *LoadResource* aufgerufen.

Instance: Bezeichnet die Instanz des Moduls, in dessen ausführbarer Datei die Ressource enthalten ist.

ResInfo: Bezeichnet die gewünschte Ressource. Dieses Handle sollte mit Hilfe der Funktion *FindResource* erzeugt werden.

Size: Bestimmt eine neudefinierte Größe in Bytes, die für die Ressource reserviert wird. Die Neudefinition wird ignoriert, wenn die Größe 0 ist.

Rückgabewert Der Rückgabewert bezeichnet den Speicherblock, der für die Ressource reserviert wurde.

Querverweis *FindResource, LoadResource*

AllocSelector

Syntax
```
function AllocSelector(Selector: Word): Word;
```

Beschreibung Diese Funktion weist einen neuen Selektor zu. Enthält der Parameter *Selector* einen gültigen Selektor, gibt *AllocSelector*

eine genaue Kopie dieses Selektors als neuen Selektor zurück. Wenn *Selector* 0 ist, wird ein neuer, nichtinitialisierter Selektor zurückgegeben.

Anwendungen müssen den neuen Selektor durch Aufruf der Funktion *FreeSelector* freigeben.

Selector: Bestimmt entweder den zu kopierenden Selektor oder ist 0, wenn *AllocSelector* einen neuen, nichtinitialisierten Selektor zuweisen soll.

Rückgabewert Der Rückgabewert ist entweder die Kopie eines bereits existierenden Selektors oder ein neuer, nichtinitialisierter Selektor. Konnte die Funktion keinen neuen Selektor zuweisen, ist der Rückgabewert 0.

Hinweis Der Aufruf von *AllocSelector* ermöglicht es Anwendungen, den zugewiesenen Selektor an die Funktion *ChangeSelector* zu übergeben.

Anwendungen sollten diese Funktion nur in Ausnahmefällen verwenden, ihre Verwendung verletzt alle empfohlenen Praktiken der Windows-Programmierung.

AnimatePalette

Syntax
```
procedure AnimatePalette(Palette: HPalette; StartIndex,
NumEntries: Word; var PaletteColors);
```

Beschreibung Diese Prozedur ersetzt Einträge in der durch den Parameter *Palette* bezeichneten virtuellen Farbpalette. Ruft eine Anwendung die Prozedur *AnimatePalette* auf, ist eine Aktualisierung des Client-Bereichs nicht nötig, da Windows die neuen Einträge unmittelbar in die Systemfarbpalette abbildet.

Palette: Bezeichnet die virtuelle Farbpalette.

StartIndex: Bestimmt den ersten zu verändernden Eintrag in der Farbpalette.

NumEntries: Bestimmt die Anzahl der zu verändernden Einträge in der Farbpalette.

PaletteColors: Zeigt auf das erste Element eines Arrays von *TPaletteEntry*-Datenstrukturen, durch die die Einträge ersetzt werden sollen. Die Einträge werden durch *StartIndex* und *NumEntries* identifiziert.

Hinweis *AnimatePalette* ändert Einträge nur, wenn im zugehörigen Feld *palPalEntry* der *TLogPalette*-Datenstruktur das Flag *pc_Reserved* gesetzt ist. *TLogPalette* definiert die aktuelle virtuelle Farbpalette. Die Funktion *CreatePalette* erzeugt eine virtuelle Farbpalette.

Querverweis *CreatePalette*

AnsiLower

Syntax
```
function AnsiLower(Str: PChar): PChar;
```

Beschreibung Diese Funktion konvertiert den vorgegebenen String in Kleinbuchstaben. Die Konvertierung wird durch den Sprachtreiber auf Grundlage der aktuell ausgewählten Sprache durchgeführt. Die Auswahl der Sprache wird vom Anwender im Setup oder in der Systemsteuerung vorgenommen.

Str: Zeigt auf einen nullterminierten String oder bestimmt ein einzelnes Zeichen. Bestimmt *Str* ein einzelnes Zeichen, liegt es im niederwertigen Byte des niederwertigen Word, das höherwertige Word ist 0.

Rückgabewert Wenn der Funktionsparameter ein String ist, zeigt der Rückgabewert auf den konvertierten String. Andernfalls ist der Rückgabewert ein 32-Bit-Wert, der das konvertierte Zeichen im niederwertigen Byte des niederwertigen Word enthält.

AnsiLowerBuff

Syntax
```
function AnsiLowerBuff(Str: PChar; Length: Word): Word;
```

Beschreibung Diese Funktion konvertiert den String im Puffer in Kleinbuchstaben. Die Konvertierung wird durch den Sprachtreiber auf Grundlage der aktuell ausgewählten Sprache durchgeführt. Die Auswahl der Sprache wird vom Anwender im Setup oder in der Systemsteuerung vorgenommen.

Str: Zeigt auf einen Puffer, der ein oder mehr Zeichen enthält.

Length: Bestimmt die Anzahl der Zeichen im durch den Parameter *Str* bezeichneten Puffer. Ist *Length* 0, beträgt die Länge 64 KByte.

Rückgabewert Der Rückgabewert bestimmt die Länge des konvertierten Strings.

AnsiNext

Syntax
```
function AnsiNext(CurrentChar: PChar): PChar;
```

Beschreibung Diese Funktion positioniert auf das nächste Zeichen in einem String.

CurrentChar: Zeigt auf ein Zeichen in einem nullterminierten String.

Rückgabewert Der Rückgabewert zeigt auf das nächste Zeichen im String. Existiert kein weiteres Zeichen, zeigt der Rückgabewert auf das Schlußzeichen des Strings.

Hinweis *AnsiNext* wird für das Bewegen durch Strings verwendet, deren Zeichen aus mindestens zwei Bytes bestehen, also beispielsweise Strings, deren Zeichen dem japanischen Zeichensatz entnommen sind.

AnsiPrev

Syntax
```
function AnsiPrev (Start, CurrentChar: PChar): PChar;
```

Beschreibung Die Funktion positioniert auf das vorhergehende Zeichen in einem String.

Start: Zeigt auf den Anfang des Strings.

CurrentChar: Zeigt auf ein Zeichen in einem nullterminierten String.

Rückgabewert Der Rückgabewert zeigt auf das vorhergehende Zeichen in einem String. Ist der Parameter *CurrentChar* identisch mit dem Parameter *Start,* zeigt der Rückgabewert auf das erste Zeichen des Strings.

Hinweis *AnsiPrev* wird für das Bewegen durch Strings verwendet, deren Zeichen aus mindestens zwei Bytes bestehen, also beispielsweise Strings, deren Zeichen dem japanischen Zeichensatz entnommen sind.

AnsiToOem

Syntax
```
function AnsiToOem(AnsiStr, OemStr: PChar): Integer;
```

Beschreibung Diese Funktion übersetzt den durch *AnsiStr* bezeichneten String vom ANSI-Zeichensatz in den OEM-Zeichensatz. Der String kann länger als 64 KByte sein.

AnsiStr: Zeigt auf einen nullterminierten String aus dem ANSI-Zeichensatz.

OemStr: Zeigt auf das Ziel, in das der String kopiert wird. Sind *OemStr* und *AnsiStr* gleich, wird der String an Ort und Stelle übersetzt.

Rückgabewert Der Rückgabewert ist immer -1.

AnsiToOemBuff

Syntax
```
procedure AnsiToOemBuff(AnsiStr, OemStr: PChar; Length:
Integer);
```

Beschreibung Diese Prozedur übersetzt den String im durch *AnsiStr* bezeichneten Puffer vom ANSI-Zeichensatz in den OEM-Zeichensatz.

AnsiStr: Zeigt auf einen Puffer, der ein oder mehr Zeichen des ANSI-Zeichensatzes enthält.

OemStr: Zeigt auf das Ziel, in das der String kopiert wird. Sind *OemStr* und *AnsiStr* gleich, wird der String an Ort und Stelle übersetzt.

Length: Bestimmt die Anzahl der Zeichen im durch den Parameter *AnsiStr* bezeichneten Puffer. Ist *Length* 0, beträgt die Länge 64 KByte.

AnsiUpper

Syntax
```
function AnsiUpper(Str: PChar): PChar;
```

Beschreibung Diese Funktion konvertiert den vorgegebenen String in Großbuchstaben.

Die Konvertierung wird durch den Sprachtreiber auf Grundlage der aktuell ausgewählten Sprache durchgeführt. Die Auswahl der Sprache wird vom Anwender im Setup oder in der Systemsteuerung vorgenommen.

Str: Zeigt auf einen nullterminierten String oder bestimmt ein einzelnes Zeichen. Bestimmt *Str* ein einzelnes Zeichen, liegt es im niederwertigen Byte des niederwertigen Word, das höherwertige Word ist 0.

Rückgabewert Wenn der Funktionsparameter ein String ist, zeigt der Rückgabewert auf den konvertierten String. Andernfalls ist der Rückgabewert ein 32-Bit-Wert, der das konvertierte Zeichen im niederwertigen Byte des niederwertigen Word enthält.

AnsiUpperBuff

Syntax
```
function AnsiUpperBuff(Str: PChar; Length: Word): Word;
```

Beschreibung Diese Funktion konvertiert den String im Puffer in Großbuchstaben.

Die Konvertierung wird durch den Sprachtreiber auf Grundlage der aktuell ausgewählten Sprache durchgeführt. Die Auswahl der Sprache wird vom Anwender im Setup oder in der Systemsteuerung vorgenommen.

Str: Zeigt auf einen Puffer, der ein oder mehr Zeichen enthält.

Length: Bestimmt die Anzahl der Zeichen im durch den Parameter *Str* bezeichneten Puffer. Ist *Length* 0, beträgt die Länge 64 KByte.

Rückgabewert Der Rückgabewert bestimmt die Länge des konvertierten Strings.

AnyPopup

Syntax
```
function AnyPopup: Bool;
```

Beschreibung Diese Funktion zeigt an, ob ein Pop-up-Fenster auf dem Bildschirm existiert. Die Funktion sucht den gesamten Windows-Bildschirm ab, nicht nur den Client-Bereich des Aufrufers. *AnyPopup* gibt auch dann einen Wert ungleich 0 zurück, wenn das Pop-up-Fenster völlig durch ein anderes Fenster verdeckt wird. Die Funktion hat keine Parameter.

Rückgabewert Der Rückgabewert ist *True*, wenn ein Pop-up-Fenster existiert, ansonsten *False*.

AppendMenu

Syntax
```
function AppendMenu(Menu: HMenu; Flags, IDNewItem:
Word; NewItem: PChar): Bool;
```

Beschreibung Diese Funktion fügt einen neuen Eintrag am Ende eines Menüs an. Anwendungen können den Status des Eintrags durch die Werte im Parameter *Flags* bestimmen.

Menu: Bezeichnet das zu verändernde Menü.

Flags: Enthält Informationen über den Status des hinzugefügten Menüeintrags. Er besteht aus einem oder mehreren Werten. Eine Liste der möglichen Werte findet sich in Kapitel 1 »Windows-Stile und -Konstanten« im Abschnitt »mf_Xxxx«.

IDNewItem: Bestimmt entweder die Befehls-ID des neuen Menüeintrags bzw. das Handle des Pop-up-Menüs, wenn *Flag* auf *Mf_Popup* gesetzt ist.

NewItem: Bestimmt den Inhalt des neuen Menüeintrags. Wird ein Bitmap als Eintrag verwendet, enthält *NewItem* das zugehörige Handle im niederwertigen Word. Die Interpretation des Parameters *NewItem* hängt vom Inhalt des Parameters *Flags* ab:

Flags	*NewItem*
mf_Bitmap	Enthält das Handle des Bitmap *HBitmap* im niederwertigen Word.
mf_OwnerDraw	Enthält einen 32-Bit-Wert, der zur Pflege zusätzlicher, dem Menüeintrag assoziierter Daten dient. Dieser ist für die Anwendung im Feld *itemData* der durch den Parameter *lParam* der Botschaften *wm_MeasureItem* und *wm_DrawItem* bezeichneten Struktur *TMeasureItemStruct* verfügbar. Diese Botschaften werden übergeben, wenn der Menüeintrag erstmals dargestellt oder verändert wird.

Flags	NewItem
mf_String	Enthält einen Zeiger auf einen nullterminierten String.

Rückgabewert Der Rückgabewert ist *True*, wenn die Funktion erfolgreich ausgeführt werden konnte, andernfalls ist er *False*.

Hinweis Wird ein Menü verändert (unabhängig davon, ob sich das Menü in einem dargestellten Fenster befindet oder nicht), sollte die Anwendung die Funktion *DrawMenuBar* aufrufen.

Jede der folgenden Gruppen zeigt Flags, die sich gegenseitig ausschließen und nicht gleichzeitig verwendet werden sollten:

mf_ByCommand und *mf_ByPosition*,
mf_Disabled, *mf_Enabled* und *mf_Grayed*,
mf_Bitmap, *mf_String* und *mf_OwnerDraw*,
mf_MenuBarBreak und *mf_MenuBreak*,
mf_Checked und *mf_Unchecked*

Eine Liste aller Flags, die im Parameter *Flags* gesetzt werden können, finden Sie in Kapitel 1 »Windows-Stile und -Konstanten« im Abschnitt »mf_Xxxx«.

Querverweis *DrawMenuBar, SetMenuItemBitmaps, wm_DrawItem, wm_MeasureItem*

Arc

Syntax
```
function Arc(DC: HDC; X1, Y1, X2, Y2, X3, Y3, X4, Y4:
Integer): Bool;
```

Beschreibung Diese Funktion zeichnet einen elliptischen Kreisbogen. Der Mittelpunkt des Kreisbogens entspricht dem Mittelpunkt des umgebenden Rechtecks, der durch die Punkte *(X1, Y1)* und *(X2, Y2)* festgelegt wird. Der Startpunkt des Kreisbogens wird durch *(X3, Y3)*, der Endpunkt durch *(X4, Y4)* bestimmt. Der Kreisbogen wird mit dem ausgewählten Stift im Uhrzeigersinn gezeichnet, wird aber nicht ausgemalt, da ein Kreisbogen keinen umschlossenen Bereich definiert.

DC: Bezeichnet den Gerätekontext.

X1: Bestimmt die virtuelle x-Koordinate der oberen linken Ecke des umgebenden Rechtecks.

Y1: Bestimmt die virtuelle y-Koordinate der oberen linken Ecke des umgebenden Rechtecks.

X2: Bestimmt die virtuelle x-Koordinate der unteren rechten Ecke des umgebenden Rechtecks.

Y2: Bestimmt die virtuelle y-Koordinate der unteren rechten Ecke des umgebenden Rechtecks.

X3: Bestimmt die virtuelle x-Koordinate des Startpunkts des Kreisbogens. Dieser Punkt muß nicht genau auf dem Kreisbogen liegen.

Y3: Bestimmt die virtuelle y-Koordinate des Startpunkts des Kreisbogens. Dieser Punkt muß nicht genau auf dem Kreisbogen liegen.

X4: Bestimmt die virtuelle x-Koordinate des Endpunkts des Kreisbogens. Dieser Punkt muß nicht genau auf dem Kreisbogen liegen.

Y4: Bestimmt die virtuelle y-Koordinate des Endpunkts des Kreisbogens. Dieser Punkt muß nicht genau auf dem Kreisbogen liegen.

Rückgabewert Der Rückgabewert ist *True*, wenn der Kreisbogen gezeichnet wird.

Hinweis Die Breite des Rechtecks wird aus *X2 - X1* berechnet und darf 32767 Einheiten nicht übersteigen. Diese Begrenzung gilt auch für die Höhe (*Y2 - Y1*).

ArrangeIconicWindows

Syntax
```
function ArrangeIconicWindows(Wnd: HWnd): Word;
```

Beschreibung Diese Funktion ordnet die Symbole des durch den Parameter *Wnd* bezeichneten Fensters an.

Wnd: Bezeichnet das übergeordnete Fenster.

Rückgabewert Der Rückgabewert ist die Höhe einer Symbolzeile. Er ist 0, wenn keine Symbole gefunden werden.

Hinweis Anwendungen, die eigene Symbole verwalten, rufen *ArrangeIconicWindows* auf, um sie in einem Client-Fenster anzuordnen. Mit ihr werden auch die Symbole auf dem Desktop neu angeordnet. Die Funktion *GetDesktopWindow* ruft das Fenster-Handle des Desktop-Fensters ab.

Um verkleinerte, untergeordnete MDI-Fenster in einem MDI-Client-Bereich anzuordnen, übergibt die Anwendung die Botschaft *wm_MDIIconArrange* an das MDI-Client-Fenster.

Querverweis *GetDesktopWindow*

BeginDeferWindowPos

Syntax
```
function BeginDeferWindowPos(NumWindows: Integer):
THandle;
```

Beschreibung Diese Funktion reserviert Speicher, der eine Datenstruktur für mehrere Fensterpositionen aufnehmen soll, und gibt dann das Handle dieser Struktur zurück.

Die Funktion *DeferWindowPos* informiert die Datenstruktur über die Zielposition des zu verschiebenden Fensters. Die Funktion *EndDeferWindowPos* übernimmt die in der Struktur abgelegten Daten und positioniert die Fenster sofort neu.

NumWindows: Bestimmt die anfängliche Anzahl von Fenstern, deren Positionsdaten in der Datenstruktur abgelegt werden. Die Funktion *DeferWindowPos* vergrößert die Datenstruktur, wenn dies nötig ist.

Rückgabewert Der Rückgabewert bezeichnet die Datenstruktur für die Fensterpositionen. Er ist 0, wenn die Systemressourcen nicht ausreichen, um Speicher für die Struktur zu reservieren.

Querverweis *DeferWindowsPos, EndDeferWindowPos*

BeginPaint

Syntax
```
function BeginPaint(Wnd: HWnd; var Paint:
TPaintStruct): HDC;
```

Beschreibung Diese Funktion bereitet das vorgegebene Fenster zum Zeichnen vor. Sie füllt die durch den Parameter *Paint* bezeichnete Struktur *TPaintStruct* mit entsprechenden Informationen.

Diese Struktur enthält das Handle des Gerätekontextes eines Fensters, eine *TRect*-Datenstruktur, die das kleinstmögliche den zu aktualisierenden Bereich umschließende Rechteck enthält und ein Flag, das bestimmt, ob der Hintergrund gelöscht wurde.

BeginPaint setzt die Clipping-Region des Gerätekontextes, um jeden Bereich außerhalb der zu aktualisierenden Region auszuschließen. Die zu aktualisierende Region wird durch die Funktionen *InvalidateRect* oder *InvalidateRgn* gesetzt.

Das System setzt diese Region nach jeder Größenveränderung, Verschiebung, Erzeugung oder anderen Operationen, die den Client-Bereich beeinflußen. Ist die zu aktualisierende Region zum Löschen markiert, sendet *BeginPaint* die Botschaft *wm_EraseBkgnd* an das Fenster.

Anwendungen sollten *BeginPaint* nur als Reaktion auf die Botschaft *wm_Paint* aufrufen. Jedem Aufruf von *BeginPaint* muß ein Aufruf der Prozedur *EndPaint* folgen.

Wnd: Bezeichnet das neu zu zeichnende Fenster.

Paint: Zeigt auf eine *TPaintStruct*-Datenstruktur, die die Zeichnungsinformationen aufnehmen soll (also den Gerätekontext, das zu aktualisierende Rechteck, usw.).

Rückgabewert Der Rückgabewert bezeichnet den Gerätekontext des vorgegebenen Fensters.

Hinweis Befindet sich das Caret im neu zu zeichnenden Bereich, wird es von *BeginPaint* versteckt, um eine Löschung zu vermeiden.

Querverweis *EndPaint, InvalidateRect, InvalidateRgn, wm_EraseBkgnd, wm_-Paint*

BitBlt

Syntax
```
function BitBlt(DestDC: HDC; X, Y, nWidth, Height:
Integer; SrcDC: HDC; XSrc, YSrc: Integer; Rop:
LongInt): Bool;
```

Beschreibung Diese Funktion bewegt ein Bitmap von dem durch *SrcDC* bezeichneten Quellgerät zu dem durch *DestDC* vorgegebenen Zielgerät. Die Parameter *XSrc* und *YSrc* bestimmen den Ursprung des Bitmap im Quellgerät. Die Parameter *X*, *Y*, *nWidth* und *Height* legen Ursprung, Breite und Höhe des durch das Bitmap auszufüllenden Rechtecks im Zielgerät fest. Der Parameter *Rop* definiert, wie die Bits von Quelle und Ziel kombiniert werden.

DestDC: Bezeichnet den Gerätekontext, der das Bitmap aufnehmen soll.

X: Bestimmt die virtuelle x-Koordinate der oberen linken Ecke des Zielrechtecks.

Y: Bestimmt die virtuelle y-Koordinate der oberen linken Ecke des Zielrechtecks.

nWidth: Bestimmt die Breite des Zielrechtecks und des Quell-Bitmaps in virtuellen Einheiten.

Height: Bestimmt die Höhe des Zielrechtecks und des Quell-Bitmaps in virtuellen Einheiten.

SrcDC: Bezeichnet den Gerätekontext, aus dem das Bitmap kopiert wird. Er muß den Wert 0 haben, wenn der Parameter *Rop* eine Rasteroperation festlegt, die keine Quelle einschließt.

XSrc: Bestimmt die virtuelle x-Koordinate der oberen linken Ecke des Quell-Bitmap.

YSrc: Bestimmt die virtuelle y-Koordinate der oberen linken Ecke des Quell-Bitmap.

Rop: Bestimmt die durchzuführende Rasteroperation.

Die Codes der Rasteroperationen legen fest, wie die grafische Geräteschnittstelle (GDI) Farben in Ausgaben kombiniert, in die der aktuelle Pinsel, ein mögliches Quell-Bitmap und ein Ziel-Bitmap einbezogen sind. Eine Liste der Codes der Raster-

operationen finden Sie in Kapitel 1 »Windows-Stile und -Konstanten« im Abschnitt »Ternäre Rasteroperationen«.

Rückgabewert Der Rückgabewert ist *True*, wenn das Bitmap gezeichnet wurde, andernfalls ist er *False*.

Hinweis Die GDI transformiert die Parameter *nWidth* und *Height*, indem sie einmal den Zielgerätekontext und einmal den Quellgerätekontext verwendet. Entsprechen die berechneten Grössen einander nicht, verwendet die GDI die Funktion *StretchBlt*, um das Quell-Bitmap auf die entsprechenden Dimensionen zu bringen. Haben Ziel-, Quell- und Vorgabe-Bitmap unterschiedliche Farbformate, konvertiert die Funktion *BitBlt* Quell- und Vorgabe-Bitmap, um sie dem Ziel anzupassen. Für die Konvertierung werden die Vorder- und Hintergrundfarben des Ziels verwendet.

Konvertiert *BitBlt* monochrome in farbige Bitmaps, so werden weiße Bits (1) auf die Hintergrundfarbe und schwarze Bits (0) auf die Vordergrundfarbe gesetzt. Hierbei werden die Vorder- und Hintergrundfarben des Zielgerätekontextes verwendet. Bei der Konvertierung von farbigen in monochrome Bitmaps setzt *BitBlt* Pixel in der Hintergrundfarbe auf Weiß (1), alle anderen auf Schwarz (0). Hierbei werden die Vorder- und Hintergrundfarben des farbigen Quellgerätekontextes verwendet.

Vordergrundfarbe ist die aktuelle Textfarbe, Hintergrundfarbe die aktuelle Hintergrundfarbe des vorgegebenen Gerätekontextes.

Nicht alle Geräte unterstützen die Funktion *BitBlt*. Weitere Informationen finden sich bei der Beschreibung der Funktion *GetDeviceCaps* und in Kapitel 1 »Windows-Stile und -Konstanten« im Abschnitt »Gerätefähigkeiten«.

Die möglichen Rasteroperationen des Parameters *Rop* finden Sie in Kapitel 1 »Windows-Stile und -Konstanten« im Abschnitt »Ternäre Rasteroperationen«.

BringWindowToTop

Syntax
```
procedure BringWindowToTop(Wnd: HWnd);
```

Beschreibung Diese Prozedur bringt ein untergeordnetes Pop-up-Fenster in den Vordergrund eines Stapels überlappender Fenster. Zusätzlich werden Pop-up-Fenster und Hauptfenster aktiviert. *BringWindowToTop* kann zur Freilegung jedes Fensters verwendet werden, das ganz oder teilweise durch überlappende Fenster verdeckt wird.

Wnd: Bezeichnet das untergeordnete Pop-up-Fenster, das in den Vordergrund gebracht werden soll.

BuildCommDCB

Syntax
```
function BuildCommDCB(Def: PChar; var DCB: TDCB):
Integer;
```

Beschreibung Diese Funktion übersetzt den durch *Def* bezeichneten Definitions-String in entsprechende Gerätesteuerblockcodes und positioniert diese Codes in dem durch *DCB* bezeichneten Block.

Def: Zeigt auf einen nullterminierten String, der Steuerinformationen für ein Gerät enthält. Der String muß das Format der Parameter des DOS-Befehls MODE haben.

DCB: Zeigt auf eine Datenstruktur *TDCB*, die den übersetzten String aufnehmen soll. Die Struktur definiert die Steuerungsvoreinstellungen eines Geräts an der seriellen Schnittstelle.

Rückgabewert Der Rückgabewert ist 0, wenn der String übersetzt wurde und negativ, wenn ein Fehler auftritt.

Hinweis *BuildCommDCB* füllt nur den Puffer. Um diese Voreinstellungen an den Anschluß weiterzuleiten, sollten Anwendungen die Funktion *SetCommState* aufrufen. Als Vorgabe stellt *BuildCommDCB* Xon/Xoff ein und deaktiviert die Hardware-Steuerung.

Außerdem sollten die Anwendungen die entsprechenden Felder in der Datenstruktur *TDCB* setzen, um die Übertragungssteuerung zu aktivieren.

Querverweis *SetCommState*

CallMsgFilter

Syntax
```
function CallMsgFilter(var Msg: TMsg; Code: Integer):
Bool;
```

Beschreibung Diese Funktion übergibt eine Botschaft und einen Code an die aktuelle Botschaftsfilterfunktion. Die Botschaftsfilterfunktion ist eine anwendungsspezifische Funktion, die alle Botschaften prüft und verändert, sie wird durch Aufruf der Funktion *SetWindowsHook* festgelegt.

Msg: Zeigt auf eine Datenstruktur *TMsg,* die die zu filternde Botschaft enthält.

Code: Gibt einen Code an, der von der Filterfunktion benutzt wird, um zu entscheiden, wie sie die Botschaft bearbeiten soll.

Rückgabewert Der Rückgabewert ist *False*, wenn die Botschaft bearbeitet werden soll und *True*, wenn die Botschaft nicht mehr bearbeitet werden soll.

Hinweis *CallMsgFilter* wird normalerweise von Windows aufgerufen, um Anwendungen die Prüfung und Kontrolle des Botschaftsflusses zu ermöglichen, wenn ein Fenster verschoben oder in der Größe verändert wird bzw. während interner Abläufe in Menüs und Bildlaufleisten.

Werte, die für den Parameter *Code* gegeben werden, dürfen sich nicht mit den von Windows an die Botschaftsfilterfunktion übergebenen *msgf_*- und *hc_*-Werten überschneiden.

Querverweis *SetWindowsHook*

CallWindowProc

Syntax
```
function CallWindowProc(PrevWndfunc: TFarProc; Wnd:
HWnd; Msg, wParam: Word; lParam: LongInt): LongInt;
```

Beschreibung Diese Funktion übergibt Botschaftsinformation an die Funktion, die durch den Parameter *PrevWndFunc* angegeben wird.

Sie wird für Fensterunterklassifizierung benutzt. Normalerweise nutzen alle Fenster mit der gleichen Klasse dieselbe Fensterfunktion gemeinsam. Eine Unterklasse ist ein Fenster oder eine Menge von Fenstern, die zur selben Fensterklasse gehören, deren Botschaften aber von einer oder mehreren anderen Funktion(en) abgefangen und bearbeitet werden, bevor sie an die Fensterfunktion dieser Klasse übergeben werden.

Die Funktion *SetWindowLong* erzeugt die Unterklasse, indem sie die einem bestimmten Fenster zugeordnete Fensterfunktion austauscht und Windows dazu veranlaßt, die neue Fensterfunktion anstelle der vorherigen aufzurufen. Jede Botschaft, die nicht von der neuen Fensterfunktion bearbeitet wird, muß an die alte Fensterfunktion durch den Aufruf von *CallWindowProc* übergeben werden. Dies erlaubt es, eine Reihe von Fensterfunktionen zu bilden.

PrevWndFunc: Ist die Adresse der Prozedurinstanz der vorherigen Fensterfunktion.

Wnd: Bezeichnet das Fenster, das die Botschaft empfängt.

Msg: Bestimmt die Botschaftsnummer.

wParam: Bestimmt zusätzliche botschaftsabhängige Informationen.

lParam: Bestimmt zusätzliche botschaftsabhängige Informationen.

Rückgabewert Es wird das Ergebnis der Bearbeitung der Botschaft zurückgeliefert. Die möglichen Rückgabewerte hängen von der gesendeten Botschaft ab.

Querverweis *SetWindowLong*

Catch

Syntax
```
function Catch(var CatchBuf: TCatchBuf): Integer;
```

Beschreibung Diese Funktion erfaßt die momentane Ausführungsumgebung und kopiert die Daten in den Puffer, auf den der Parameter *CatchBuf* zeigt. Die Ausführungsumgebung ist der Zustand aller Systemregister und des Befehlszählers.

CatchBuf: Zeigt auf die *TCatchBuf*-Struktur, die die Ausführungsumgebung aufnimmt.

Rückgabewert Der Rückgabewert ist 0, wenn die Umgebung in den Puffer kopiert wurde.

Hinweis Die Funktion *Throw* verwendet den Puffer zur Wiederherstellung der Ausführungsumgebung mit den vorherigen Werten.

Querverweis *Throw*

ChangeClipboardChain

Syntax
```
function ChangeClipboardChain(Wnd, WndNext: HWnd): Bool;
```

Beschreibung Diese Funktion entfernt das durch *Wnd* angegebene Fenster aus der Kette der Clipboard-Viewer und macht das durch *WndNext* angegebene Fenster zum Nachfolger des entfernten Fensters in der Kette.

Wnd: Bezeichnet das Fenster, das aus der Kette enfernt werden soll. Das Handle muß vorher an die Funktion *SetClipboardViewer* übergeben worden sein.

WndNext: Bezeichnet das Fenster, das *Wnd* in der Kette der Clipboard-Viewer folgt (dies ist das Handle, das von der Funktion *SetClipboardViewer* zurückgegeben wird, sofern die Reihenfolge nicht infolge einer *wm_ChangeCBChain*-Botschaft geändert wurde).

Rückgabewert Der Rückgabewert gibt den Status des *Wnd*-Fensters an. Er ist *True*, wenn das Fenster gefunden und entfernt wurde, andernfalls ist er *False*.

Querverweis *SetClipboardViewer, wm_ChangeCBChain*

ChangeMenu

Syntax
```
function ChangeMenu(A: hMenu; B: Word; C: PChar;
D: Word; E: Word): Bool;
```

Beschreibung Diese veraltete Funktion wurde durch fünf spezialisierte Funktionen ersetzt. Diese neuen Funktionen sind:

Funktion	Beschreibung
AppendMenu	Hängt eine neue Menüoption an das Ende eines Menüs an.
DeleteMenu	Löscht eine Menüoption aus einem Menü und gibt sie frei.
InsertMenu	Fügt eine Menüoption in ein Menü ein.
ModifyMenu	Modifiziert eine Menüoption in einem Menü.
RemoveMenu	Entfernt eine Menüoption aus einem Menü, gibt sie aber nicht frei.

Die Benutzung von *ChangeMenu* wird nicht empfohlen. Neue Anwendungen sollten die oben aufgelisteten neuen Funktionen verwenden.

Querverweis *AppendMenu, DeleteMenu, InsertMenu, ModifyMenu, RemoveMenu*

ChangeSelector

Syntax
```
function ChangeSelector(DestSelector, SourceSelector:
Word): Word;
```

Beschreibung Diese Funktion generiert einen Codeselektor, der einem gegebenen Datenselektor entspricht, oder einen Datenselektor, der einem gegebenen Codeselektor entspricht.

Der Parameter *SourceSelector* gibt den Selektor an, der kopiert und umgewandelt werden soll, der Parameter *DestSelector* ist ein Selektor, der vorher durch einen Aufruf der Funktion *AllocSelector* reserviert wurde.

ChangeSelector modifiziert den Zielselektor dahingehend, daß er die gleichen Eigenschaften wie der Quellselektor hat, jedoch mit umgekehrtem Code- oder Datenattribut. Diese Funktion ändert nur die Attribute, nicht aber den Wert des Selektors.

DestSelector: Gibt einen vorher durch *AllocSelector* reservierten Selektor an, der den umgewandelten Selektor aufnimmt.

SourceSelector: Gibt den Selektor an, der umgewandelt werden soll.

Rückgabewert Der Rückgabewert ist der kopierte und umgewandelte Selektor. Er ist 0, wenn ein Fehler aufgetreten ist.

Hinweis Windows nimmt keine Änderungen am Quellselektor vor. Daher sollte die Anwendung den umgewandelten Zielselektor sofort verwenden, wenn er von dieser Funktion zurückgeliefert, bevor eine eventuelle Speicherverschiebung auftreten kann.

Eine Anwendung sollte diese Funktion nur verwenden, wenn dies unbedingt notwendig ist. Der Gebrauch dieser Funktion verletzt alle gängigen Windows-Programmierpraktiken.

Querverweis *AllocDStoCSAlias*

CheckDlgButton

Syntax
```
procedure CheckDlgButton(Dlg: HWnd; IDButton: Integer;
Check: Word);
```

Beschreibung Diese Prozedur markiert einen Schalter, entfernt eine Auswahlmarkierung oder ändert den Zustand eines dreistufigen Schalters. *CheckDlgButton* sendet eine *bm_SetCheck*-Botschaft an den Schalter, der die angegebene ID im gegebenen Dialogfenster hat.

Dlg: Bezeichnet das Dialogfenster, das den Schalter enthält.

IDButton: Gibt den Schalter an, der modifiziert werden soll.

Check: Gibt die vorzunehmende Handlung an. Ist der Parameter *Check* ungleich 0, markiert die Funktion *CheckDlgButton* den Schalter, ist er 0, wird die Auswahlmarkierung entfernt. Bei dreistufigen Schaltern wird der Schalter als nicht-aktivierbarer Zustand dargestellt, wenn *Check* den Wert 2 hat, er wird markiert, wenn *Check* 1 ist und die Markierung wird entfernt, wenn *Check* 0 ist.

CheckMenuItem

Syntax
```
function CheckMenuItem(Menu: HMenu; IDCCheckItem,
Check: Word): Bool;
```

Beschreibung Diese Funktion plaziert eine Auswahlmarkierung neben einer Menüoption oder entfernt sie von einer Menüoption in dem durch den Parameter *Menu* angegebenen Pop-up-Menü. Der Parameter *IDCCheckItem* gibt die Option an, die verändert werden soll.

Menu: Bezeichnet das Menü.

IDCCheckItem: Gibt die Menüoption an, die bearbeitet werden soll.

Check: Gibt an, wie die Menüoption markiert werden soll und wie die Position der Option im Menü festgelegt werden soll.

Der Parameter *Check* kann eine Kombination der Flags *mf_Checked* oder *mf_Unchecked* mit *mf_ByPosition* oder *mf_ByCommand* sein.

Diese Flags können durch Verwendung des bitweisen **or**-Operators kombiniert werden.

Wert	Bedeutung
mf_ByCommand	Der Parameter *IDCCheckItem* bestimmt die ID der Menüoption (Voreinstellung).
mf_ByPosition	Der Parameter *IDCCheckItem* bestimmt die Position der Menüoption (die erste Menüoption liegt an Position 0).
mf_Checked	Setzt die Auswahlmarkierung.
mf_Unchecked	Entfernt die Auswahlmarkierung.

Rückgabewert

Der Rückgabewert gibt den momentanen Zustand der Option an, dieser ist entweder *mf_Checked* oder *mf_Unchecked*.

Ist die Menüoption nicht vorhanden, so ist der Rückgabewert -1.

Hinweis

Der Parameter *IDCCheckItem* kann eine Pop-up-Menüoption oder eine Menüoption bezeichnen. Es sind keinerlei speziellen Schritte erforderlich, um eine Pop-up-Menüoption zu markieren.

Hauptmenüoptionen können nicht markiert werden. Eine Pop-up-Menüoption sollte durch Übergabe ihrer Position markiert werden, da sie keinen zugeordneten Bezeichner hat.

CheckRadioButton

Syntax

```
procedure CheckRadioButton(Dlg: HWnd; IDFirstButton,
IDLastButton, IDCheckButton: Integer);
```

Beschreibung

Diese Prozedur markiert das durch den Parameter *IDCheckButton* angegebene Schaltfeld und entfernt die Auswahlmarkierung von allen anderen Schaltfeldern in der durch die Parameter *IDFirstButton* und *IDLastButton* definierten Gruppe von Schaltern.

Die Prozedur *CheckRadioButton* sendet eine *bm_SetCheck*-Botschaft an das Schaltfeld, das die angegebene ID im gegebenen Dialogfenster hat.

Dlg: Bezeichnet das Dialogfenster.

IDFirstButton: Bestimmt den Integer-Bezeichner des ersten Schaltfelds in der Gruppe.

IDLastButton: Bestimmt den Integer-Bezeichner des letzten Schaltfelds in der Gruppe.

IDCheckButton: Bestimmt den Integer-Bezeichner des Schaltfelds, das markiert werden soll.

ChildWindowFromPoint

Syntax
```
function ChildWindowFromPoint(Wnd: HWnd; Point: TPoint): HWnd;
```

Beschreibung Diese Funktion überprüft, welches der dem Fenster *Wnd* untergeordnete Fenster den angegebenen Punkt enthält.

Wnd: Bezeichnet das übergeordnete Fenster.

Point: TPoint-Struktur der Client-Koordinaten des Punktes, der getestet werden soll.

Rückgabewert Der Rückgabewert bezeichnet das untergeordnete Fenster, das den Punkt enthält. Er ist 0, wenn der übergebene Punkt außerhalb des übergeordneten Fensters liegt. Wenn der Punkt innerhalb des übergeordneten Fensters, aber nicht in einem der untergeordneten Fenster liegt, wird das Handle des übergeordneten Fensters zurückgegeben.

Chord

Syntax
```
function Chord(DC: HDC; X1, Y1, X2, Y2, X3, Y3, X4, Y4: Integer): Bool;
```

Beschreibung Diese Funktion zeichnet einen elliptischen Bogen (ein Bereich, der durch den Schnitt einer Ellipse und eines Liniensegments eingegrenzt wird). Die Parameter (*X1*, *Y1*) und (*X2*, *Y2*) geben die obere linke bzw. untere rechte Ecke eines Rechtecks an, das die Ellipse, die ein Teil des Bogens ist, umgibt. Die Parameter (*X3*, *Y3*) und (*X4*, *Y4*) geben die Endpunkte einer Linie an, die die Ellipse schneidet.

DC: Bezeichnet den Gerätekontext, in dem der elliptische Bogen erscheinen wird.

X1: Bestimmt die x-Koordinate der oberen linken Ecke des eingrenzenden Rechtecks.

Y1: Bestimmt die y-Koordinate der oberen linken Ecke des eingrenzenden Rechtecks.

X2: Bestimmt die x-Koordinate der unteren rechten Ecke des eingrenzenden Rechtecks.

Y2: Bestimmt die y-Koordinate der unteren rechten Ecke des eingrenzenden Rechtecks.

X3: Bestimmt die x-Koordinate des ersten Endpunkts des Liniensegments.

Y3: Bestimmt die y-Koordinate des ersten Endpunkts des Liniensegments.

X4: Bestimmt die x-Koordinate des zweiten Endpunkts des Liniensegments.

Y4: Bestimmt die y-Koordinate des zweiten Endpunkts des Liniensegments.

Rückgabewert Der Rückgabewert ist *True*, wenn der Bogen gezeichnet wurde, andernfalls ist er *False*.

ClearCommBreak

Syntax
```
function ClearCommBreak(Cid: Integer): Integer;
```

Beschreibung Diese Funktion stellt die zeichenweise Übertragung wieder her und versetzt die Übertragungsleitung in den nicht-unterbrochenden Zustand.

Cid: Gibt die Schnittstelle an, die angesprochen werden soll. Die Funktion *OpenComm* liefert diesen Wert zurück.

Rückgabewert Der Rückgabewert ist 0, wenn die Funktion erfolgreich ausgeführt wurde.

Er ist negativ, wenn der Parameter *Cid* keine gültige Geräteeinheit bezeichnet.

Querverweis *OpenComm*

ClientToScreen

Syntax
```
procedure ClientToScreen(Wnd: HWnd; var Point: TPoint);
```

Beschreibung Diese Prozedur wandelt die Client-Koordinaten eines gegebenen Punktes auf dem Bildschirm in Bildschirmkoordinaten um.

ClientToScreen benutzt die Client-Koordinaten der *TPoint*-Datenstruktur des Parameters *Point*, um neue Bildschirmkoordinaten zu errechnen, danach ersetzt sie die Koordinaten in der Struktur durch die neuen Koordinaten. Die neuen Bildschirmkoordinaten sind relativ zur oberen linken Ecke der Systemanzeige.

Wnd: Bezeichnet das Fenster, dessen Client-Bereich für die Umwandlung verwendet wird.

Point: Eine *TPoint*-Datenstruktur, die die umzuwandelnden Client-Koordinaten enthält.

Hinweis *ClientToScreen* setzt voraus, daß der übergebene Punkt in Client-Koordinaten und relativ zum vorgegebenen Fenster angegeben wird.

ClipCursor

Syntax
```
procedure ClipCursor(Rect: PRect);
```

Beschreibung Diese Prozedur schränkt den Mauszeiger auf das durch den Parameter *Rect* übergebene Rechteck auf dem Bildschirm ein. Liegt eine spätere Mauszeigerposition, die durch die Funktion *SetCursorPos* oder die Maus vorgegeben wird, außerhalb des Rechtecks, paßt Windows diese Position automatisch an, um den Mauszeiger innerhalb des Rechtecks zu halten. Ist *Rect* **nil**, kann sich der Mauszeiger auf dem ganzen Bildschirm bewegen.

Rect: Zeigt auf eine *TRect*-Datenstruktur, die die Bildschirmkoordinaten der oberen linken und der unteren rechten Ecke des eingrenzenden Rechtecks enthält.

Hinweis Der Mauszeiger ist eine gemeinsam genutzte Ressource. Eine Anwendung, die den Mauszeiger auf ein gegebenes Rechteck begrenzt hat, muß ihn wieder freigeben, bevor sie die Steuerung einer anderen Anwendung überläßt.

Querverweis *SetCursorPos*

CloseClipboard

Syntax
```
function CloseClipboard: Bool;
```

Beschreibung Diese Funktion schließt die Zwischenablage. Sie sollte aufgerufen werden, wenn ein Fenster das Prüfen oder Ändern der Zwischenablage beendet hat. Sie ermöglicht dadurch anderen Anwendungen den Zugriff auf die Zwischenablage.

Rückgabewert Der Rückgabewert ist *True*, wenn die Zwischenablage geschlossen wurde, andernfalls ist er *False*.

CloseComm

Syntax
```
function CloseComm(Cid: Integer): Integer;
```

Beschreibung Diese Funktion schließt die durch den Parameter *Cid* angegebene Schnittstelle und gibt den Speicher frei, der für die Übertragungs- und Empfangs-Warteschlangen der Geräteeinheit belegt wurde. Alle Zeichen in der Ausgabe-Warteschlange werden gesendet, bevor die Schnittstelle geschlossen wird.

Cid: Gibt die Geräteeinheit an, die geschlossen werden soll. Die Funktion *OpenComm* gibt diesen Wert zurück.

Rückgabewert Der Rückgabewert ist 0, wenn die Schnittstelle geschlossen wurde und negativ, wenn ein Fehler aufgetreten ist.

Querverweis *OpenComm*

CloseMetaFile

Syntax
```
function CloseMetaFile(DC: THandle): THandle;
```

Beschreibung Diese Funktion schließt den Metafile-Gerätekontext und erzeugt ein Metafile-Handle, das verwendet werden kann, um das Metafile mit der Funktion *PlayMetaFile* abzuspielen.

DC: Bezeichnet den Metafile-Gerätekontext, der geschlossen werden soll.

Rückgabewert Der Rückgabewert bezeichnet das Metafile, wenn die Funktion erfolgreich war, andernfalls ist er 0.

Querverweis *PlayMetaFile*

CloseSound

Syntax
```
procedure CloseSound;
```

Beschreibung Diese Prozedur schließt den Zugriff auf den Lautsprecher und gibt die Geräteeinheit für das Öffnen durch andere Anwendungen frei. Sie löscht alle Warteschlangen und gibt jeden hierfür reservierten Puffer frei. Diese Prozedur hat keine Parameter.

CloseWindow

Syntax
```
procedure CloseWindow(Wnd: HWnd);
```

Beschreibung Diese Prozedur verkleinert das angegebene Fenster. Wenn das Fenster andere überlappt, werden Client-Bereich und Überschrift des geöffneten Fensters vom Bildschirm entfernt und das Symbol in den Symbolbereich des Bildschirms verschoben.

Wnd: Bezeichnet das Fenster, das verkleinert werden soll.

Hinweis Diese Prozedur hat keine Wirkung, wenn der Parameter *Wnd* das Handle eines Pop-up- oder untergeordneten Fensters ist.

CombineRgn

Syntax
```
function CombineRgn(DestRgn, SrcRgn1, SrcRgn2: HRgn;
CombineMode: Integer): Integer;
```

Beschreibung Diese Funktion erzeugt eine neue Region, indem sie zwei existierende Regionen auf die durch den Parameter *CombineMode* angegebe Weise verbindet.

DestRgn: Bezeichnet eine Region, die durch die neue Region ersetzt wird.

SrcRgn1: Bezeichnet eine Region.

SrcRgn2: Bezeichnet eine Region.

CombineMode: Gibt die Operation an, die auf die zwei bestehenden Regionen angewandt werden soll. Er kann jeden der folgenden Werte annehmen:

Wert	Bedeutung
rgn_And	Verwendet nur die überlappenden Bereiche beider Regionen (Schnittmenge).
rgn_Copy	Erzeugt eine Kopie von Region 1 (*SrcRgn1).*
rgn_Diff	Speichert die Bereiche von Region 1 (*SrcRgn1*), die nicht Teil von Region 2 (*SrcRgn2*) sind.
rgn_Or	Verbindet beide Regionen (Vereinigung).
rgn_Xor	Verbindet beide Regionen, entfernt aber überlappende Bereiche.

Rückgabewert

Der Rückgabewert bestimmt den Typ der entstandenen Region. Folgende Werte sind möglich:

Wert	Bedeutung
ComplexRegion	Neue Region hat überlappende Ränder.
Error	Keine neue Region erzeugt.
NullRegion	Neue Region ist leer.
SimpleRegion	Neue Region hat keine überlappenden Ränder.

CopyMetaFile

Syntax

```
function CopyMetaFile(SrcMetaFile: THandle; FileName:
PChar): THandle;
```

Beschreibung

Diese Funktion kopiert das Quell-Metafile in die Datei, auf die der Parameter *FileName* zeigt und gibt ein Handle für das neue Metafile zurück. Wenn *FileName* 0 ist, wird die Quelldatei in ein Speicher-Metafile kopiert.

SrcMetaFile: Bezeichnet das Quell-Metafile.

FileName: Zeigt auf einen mit 0 abgeschlossenen String, der die Datei angibt, die das Metafile aufnehmen soll.

Rückgabewert

Der Rückgabewert bezeichnet das neue Metafile.

CopyRect

Syntax

```
procedure CopyRect(var DestRect, SourceRect: TRect);
```

Beschreibung

Diese Prozedur kopiert das Rechteck, auf das der Parameter *SourceRect* zeigt, in die *TRect*-Datenstruktur, auf die der Parameter *DestRect* zeigt.

DestRect: Eine *TRect*-Datenstruktur.

SourceRect: Eine *TRect*-Datenstruktur.

CountClipboardFormats

Syntax
```
function CountClipboardFormats: Integer;
```

Beschreibung Diese Funktion liefert die Anzahl der Formate, die von der Zwischenablage verarbeitet werden können.

Die Funktion hat keine Parameter.

Rückgabewert Der Rückgabewert gibt die Anzahl von Datenformaten in der Zwischenablage an.

CountVoiceNotes

Syntax
```
function CountVoiceNotes(Voice: Integer): Integer;
```

Beschreibung Diese Funktion liefert die Anzahl der Noten in der angegebenen Klang-Warteschlange. Es werden nur diejenigen Einträge in der Warteschlange gezählt, die aus einem Aufruf der Funktion *SetVoiceNote* hervorgegangen sind.

Voice: Gibt die Klang-Warteschlange an, deren Noten gezählt werden sollen. Die Nummer der ersten Warteschlange ist 1.

Rückgabewert Der Rückgabewert gibt die Anzahl der Noten in der gegebenen Klang-Warteschlange an.

Querverweis *SetVoiceNote*

CreateBitmap

Syntax
```
function CreateBitmap(Width, Height: Integer; Planes,
BitCount: Byte; Bits: Pointer): HBitmap;
```

Beschreibung Diese Funktion erzeugt ein geräteabhängiges Bitmap, das die angegebene Breite, Höhe und Bitmuster hat. Das Bitmap kann später als das aktuelle Bitmap ausgewählt werden, indem man die Funktion *SelectObject* verwendet.

Obwohl ein Bitmap nicht direkt in ein Anzeigegerät kopiert werden kann, kann *BitBlt* es aus einem Speicheranzeigekontext (in dem es das aktuelle Bitmap ist) in jedes andere kompatible Gerät kopieren.

Width: Bestimmt die Breite des Bitmap (in Pixel).

Height: Bestimmt die Höhe des Bitmap (in Pixel).

Planes: Bestimmt die Anzahl der Farbebenen im Bitmap. Jede Ebene besitzt *Width * Height * BitCount* Bits.

BitCount: Bestimmt die Anzahl von Farbbits je Bildschirmpixel.

Bits: Zeigt auf ein Byte-Array, das die ursprünglichen Bitwerte des Bitmap enthält. Wenn er 0 ist, wird das neue Bitmap nicht initialisiert. Weitere Informationen finden Sie in der Beschreibung des Feldes *bmBits* der *TBitmap*-Datenstruktur.

Rückgabewert Wenn die Funktion erfolgreich ausgeführt wurde, bezeichnet der Rückgabewert ein Bitmap, andernfalls ist er 0.

Querverweis *BitBlt, SelectObject*

CreateBitmapIndirect

Syntax
```
function CreateBitmapIndirect(var Bitmap: TBitmap):
HBitmap;
```

Beschreibung Diese Funktion erzeugt ein Bitmap, dessen Breite, Höhe und Bitmuster durch die Datenstruktur, auf die der Parameter *Bitmap* zeigt, vorgegeben werden.

Obwohl ein Bitmap nicht direkt für einen Bildschirm ausgewählt werden kann, kann es als aktuelles Bitmap gewählt und in jedes kompatible Anzeigegerät durch Verwendung der Funktion *BitBlt* kopiert werden.

Bitmap: Eine *TBitmap*-Datenstruktur, die Informationen über das Bitmap enthält.

Rückgabewert Wenn die Funktion erfolgreich ausgeführt wurde, bezeichnet der Rückgabewert ein Bitmap, andernfalls ist er 0.

Querverweis *BitBlt*

CreateBrushIndirect

Syntax
```
function CreateBrushIndirect(var LogBrush: TLogBrush):
HBrush;
```

Beschreibung Diese Funktion erzeugt einen virtuellen Pinsel, dessen Stil, Farbe und Muster durch die Datenstruktur vorgegeben werden, auf die der Parameter *LogBrush* zeigt. Der Pinsel kann später für jede Geräteeinheit als aktueller Pinsel gewählt werden.

LogBrush: Eine *TLogBrush*-Datenstruktur, die Informationen über den Pinsel enthält.

Rückgabewert Wenn die Funktion erfolgreich ausgeführt wurde, bezeichnet der Rückgabewert einen virtuellen Pinsel, andernfalls ist er 0.

Hinweis Ein Pinsel, der unter Verwendung eines monochromen (eine Ebene, ein Bit pro Pixel) Bitmap erzeugt wurde, wird mit der aktuellen Text- und Hintergrundfarbe dargestellt. Pixel, die durch ein auf 0 gesetztes Bit repräsentiert werden, werden in

der aktuellen Textfarbe gezeichnet. Pixel, bei denen das Bit auf 1 gesetzt wird, werden in der aktuellen Hintergrundfarbe gezeichnet.

CreateCaret

Syntax

```
procedure CreateCaret(Wnd: HWnd; Bitmap: HBitmap;
Width, Height: Integer);
```

Beschreibung

Diese Prozedur erzeugt eine neue Form für das System-Caret und weist dem gegebenen Fenster den Besitz des Caret zu. Die Form des Caret kann eine Linie, ein Block oder ein Bitmap sein. Sie wird durch den Parameter *Bitmap* bestimmt.

Wenn *Bitmap* ein Bitmap-Handle ist, werden die Parameter *Width* und *Height* ignoriert, das Bitmap bestimmt seine eigene Breite und Höhe. (Das Bitmap-Handle muß vorher durch die Verwendung einer der Funktionen *CreateBitmap*, *CreateDIBitmap* oder *LoadBitmap* erzeugt worden sein.)

Wenn *Bitmap* 0 oder 1 ist, bestimmen *Width* und *Height* die Breite und Höhe (in virtuellen Einheiten) des Caret, die genaue Breite und Höhe (in Pixel) hängen vom Abbildungsmodus des Fensters ab.

Wenn *Width* oder *Height* 0 ist, richten sich Breite oder Höhe des Caret nach der Breite oder Höhe des Fensterrandes des Systems. Die Verwendung der Breite oder der Höhe des Fensterrandes garantiert, daß das Caret auf einem hochauflösenden Bildschirm sichtbar ist.

CreateCaret entfernt eine eventuell vorhergehende Caret-Form automatisch, ohne Rücksicht darauf zu nehmen, welches Fenster das Caret besitzt. Wenn das Caret erzeugt ist, ist es zunächst unsichtbar, um es sichtbar zu machen, muß die Funktion *ShowCaret* aufgerufen werden.

Wnd: Bezeichnet das Fenster, das das neue Caret besitzt.

Bitmap: Bezeichnet das Bitmap, das die Form des Caret festlegt. Wenn *Bitmap* 0 ist, ist das Caret schwarz; wenn *Bitmap* 1 ist, ist es grau.

Width: Bestimmt die Breite des Caret in virtuellen Einheiten.

Height: Bestimmt die Höhe des Caret in virtuellen Einheiten.

Hinweis

Das System-Caret ist eine gemeinsam genutzte Ressource. Ein Fenster sollte ein Caret nur dann erzeugen, wenn es den Fokus hat oder aktiv ist. Es sollte das Caret wieder entfernen, bevor es den Fokus verliert oder inaktiv wird.

Die Breite oder Höhe des Systemfensterrandes kann durch Verwendung der Funktion *GetSystemMetrics* mit den Codes *sm_CXBorder* und *sm_CYBorder* ermittelt werden (vergleichen

Sie hierzu den Abschnitt »sm_Xxxx« in Kapitel 1 »Windows-Stile und -Konstanten«, .

Querverweis *CreateBitmap, CreateDIBitmap, GetSystemMetrics, LoadBitmap*

CreateCompatibleBitmap

Syntax
```
function CreateCompatibleBitmap(DC: HDC; Width, Height:
Integer): HBitmap;
```

Beschreibung Diese Funktion erzeugt ein Bitmap, das mit dem durch den Parameter *DC* angegebenen Gerätekontext kompatibel ist. Das Bitmap hat die gleiche Anzahl an Farbebenen oder das gleiche Bits-pro-Pixel-Format wie der angegebene Gerätekontext. Es kann als aktuelles Bitmap für jedes Speichergerät, das mit dem von *DC* angegebenen kompatibel ist, gewählt werden.

Wenn *DC* ein Speichergerätekontext ist, hat das zurückgelieferte Bitmap das gleiche Format wie das in diesem Gerätekontext momentan gewählte Bitmap. Ein Speichergerätekontext ist ein Speicherblock, der eine Bildschirmoberfläche darstellt. Er kann dazu benutzt werden, Bilder im Speicher vorzubereiten, bevor sie auf die eigentliche Bildschirmoberfläche der kompatiblen Geräteeinheit kopiert werden.

Wenn ein Speichergerätekontext erzeugt wird, wählt die GDI automatisch ein vordefiniertes monochromes Bitmap.

Da ein farbiger Speichergerätekontext sowohl farbige als auch monochrome Bitmaps ausgewählt haben kann, ist das Format des Bitmap, das von *CreateCompatibleBitmap* zurückgeliefert wird, nicht immer das gleiche. Das Format eines kompatiblen Bitmap für einen Gerätekontext, der kein Speicher ist, entspricht immer dem Format der Geräteeinheit.

DC: Bezeichnet den Gerätekontext.

Width: Bestimmt die Breite des Bitmap (in Bits).

Height: Bestimmt die Höhe des Bitmap (in Bits).

Rückgabewert Der Rückgabewert bezeichnet ein Bitmap, wenn die Funktion erfolgreich war, andernfall ist er 0.

CreateCompatibleDC

Syntax
```
function CreateCompatibleDC(DC: HDC): HDC;
```

Beschreibung Diese Funktion erzeugt einen Speichergerätekontext, der kompatibel mit der durch den Parameter *DC* bestimmten Geräteeinheit ist. Ein Speichergerätekontext ist ein Speicherblock, der eine Bildschirmoberfläche darstellt. Er kann dazu benutzt werden, Bilder im Speicher vorzubereiten, bevor sie auf die eigentliche Bilschirmoberfläche der kompatiblen Geräteein-

heit kopiert werden. Wenn ein Speichergerätekontext erzeugt wird, wählt die GDI für ihn automatisch ein vordefiniertes monochromes 1-zu-1 Bitmap.

DC: Bezeichnet den Gerätekontext. Wenn *DC* 0 ist, erzeugt die Funktion einen Speichergerätekontext, der kompatibel zum Systembildschirm ist.

Rückgabewert Wenn die Funktion erfolgreich ausgeführt wurde, bezeichnet der Rückgabewert den neuen Speichergerätekontext, andernfalls ist er 0.

Hinweis Diese Funktion kann nur dazu benutzt werden, kompatible Gerätekontexte für Geräteeinheiten, die Rasteroperationen unterstützen, zu erzeugen.

Weitere Informationen finden Sie bei der Beschreibung der Funktion *GetDeviceCaps* und in Kapitel 1 »Windows-Stile und -Konstanten« im Abschnitt »dc_Xxxx«.

GDI-Ausgabefunktionen können mit einem Speichergerätekontext nur dann benutzt werden, wenn ein Bitmap in diesem Kontext erzeugt und ausgewählt wurde.

Wenn eine Anwendung den Gerätekontext nicht länger benötigt, sollte sie ihn freigeben, indem sie die Funktion *DeleteDC* aufruft.

Querverweis *DeleteDC, GetDeviceCaps*

CreateCursor

Syntax
```
function CreateCursor(Instance: THandle; Xhotspot,
Yhotspot, Width, Height: Integer; ANDbitPlane,
XORbitPlane: Pointer): HCursor;
```

Beschreibung Diese Funktion erzeugt einen Mauszeiger mit der vorgegebenen Breite, Höhe und dem vorgegebenen Bitmuster.

Instance: Bezeichnet eine Instanz des Moduls, das den Mauszeiger erzeugt.

Xhotspot: Bestimmt die horizontale Position des Aktionspunktes des Mauszeigers.

Yhotspot: Bestimmt die vertikale Position des Aktionspunktes des Mauszeigers.

Width: Bestimmt die Breite des Mauszeigers in Pixeln.

Height: Bestimmt die Höhe des Mauszeigers in Pixeln.

ANDbitPlane: Zeigt auf ein Byte-Array, das die Bitwerte für die **and**-Maske des Mauszeigers enthält.

Dies können die Bits eines geräteabhängigen monochromen Bitmaps sein.

XORbitPlane: Zeigt auf ein Byte-Array, das die Bitwerte für die **xor**-Maske des Mauszeigers enthält. Dies können die Bits eines geräteabhängigen monochromen Bitmaps sein.

Rückgabewert Wenn die Funktion erfolgreich ausgeführt wurde, bezeichnet der Rückgabewert den Mauszeiger, andernfalls ist er 0.

CreateDC

Syntax
```
function CreateDC(DriverName, DeviceName, Output:
PChar; InitData: Pointer): HDC;
```

Beschreibung Diese Funktion erzeugt einen Gerätekontext für die angegebene Geräteeinheit. Die Parameter *DriverName*, *DeviceName* und *Output* geben den Gerätetreiber, den Gerätenamen und das physikalische Ausgabemedium an (Datei oder Anschluß).

DriverName: Zeigt auf einen nullterminierten String, der den DOS-Dateinamen (ohne Namenserweiterung) des Gerätetreibers angibt (zum Beispiel »Epson«).

DeviceName: Zeigt auf einen mit 0 abgeschlossenen String, der den Namen der Geräteeinheit angibt, die unterstützt werden soll (zum Beispiel »Epson FX-80«). Der Parameter *DeviceName* wird verwendet, wenn das Modul mehr als eine Geräteeinheit unterstützt.

Output: Zeigt auf einen mit 0 abgeschlossenen String, der den DOS-Dateinamen oder den Gerätenamen für das physikalische Ausgabemedium enthält (Datei oder Ausgabeanschluß).

InitData: Zeigt auf eine *TDevMode*-Datenstruktur, die gerätespezifische Initialisierungsdaten für den Gerätetreiber enthält. *ExtDeviceMode* ruft diese Struktur ab, die für eine gegebene Geräteeinheit gefüllt ist. Der Parameter *InitData* muß 0 sein, wenn der Gerätetreiber die eventuell vorhandene voreingestellte Initialisierung, die vom Anwender durch die Systemsteuerung angegeben wird, verwenden soll.

Rückgabewert Wenn die Funktion erfolgreich war, bezeichnet der Rückgabewert einen Gerätekontext für die angegebene Geräteeinheit, andernfalls ist er 0.

Hinweis DOS-Gerätenamen folgen den DOS-Konventionen, ein Doppelpunkt (:) am Ende ist empfehlenswert, jedoch optional. Windows entfernt den abschließenden Doppelpunkt, so daß ein Gerätename, der mit einem Doppelpunkt endet, dem gleichen Anschluß zugeordnet wird wie derselbe Name ohne einen Doppelpunkt. Die Treiber- und Anschlußnamen dürfen keine vorangestellten oder nachfolgende Leerzeichen enthalten.

CreateDialog

Syntax

```
function (Instance: THandle; TemplateName: PChar;
WndParent: HWnd; DialogFunc: TFarProc): HWnd;
```

Beschreibung

Diese Funktion erzeugt ein nichtmodales Dialogfenster, das die Größe, den Stil und die Steuerelemente besitzt, die durch die vom Parameter *TemplateName* vorgegebene Dialogfenstervorlage definiert werden. Der Parameter *WndParent* bezeichnet ein das Dialogfenster besitzendes Anwendungsfenster. Die Dialogfunktion, auf die der Parameter *DialogFunc* zeigt, bearbeitet jede Botschaft, die vom Dialogfenster empfangen wird.

CreateDialog sendet eine *wm_InitDialog*-Botschaft an die Dialogfunktion, bevor sie das Dialogfenster anzeigt. Diese Botschaft erlaubt der Dialogfunktion, die Steuerelemente des Dialogfensters zu initialisieren.

CreateDialog kehrt sofort zurück, wenn sie das Dialogfenster erzeugt hat, sie wartet nicht, bis das Dialogfenster die Eingabebearbeitung beginnt.

Instance: Bezeichnet eine Instanz des Moduls, dessen ausführbare Datei die Vorlage des Dialogfensters enthält.

TemplateName: Zeigt auf einen nullterminierten String, der die Vorlage des Dialogfensters benennt.

WndParent: Bezeichnet das Fenster, welches das Dialogfenster besitzt.

DialogFunc: Ist die Adresse der Prozedurinstanz der Dialogfunktion. Details entnehmen Sie dem folgenden Hinweis.

Rückgabewert

Bei erfolgreicher Ausführung der Funktion ist der Rückgabewert das Fenster-Handle des Dialogfensters, er ist 0, wenn die Funktion das Dialogfenster nicht erzeugen kann.

Hinweis

Verwenden Sie den Stil *ws_Visible* in der Vorlage des Dialogfensters, wenn das Dialogfenster nach der Erzeugung im übergeordneten Fenster erscheinen soll.

Verwenden Sie die Funktion *DestroyWindow*, um ein Dialogfenster zu entfernen, das durch *CreateDialog* erzeugt wurde.

Ein Dialogfenster kann bis zu 255 Steuerelemente haben.

Callback-Funktion

```
function DialogFunc (hdlg: HWnd; wMsg, wParam: Word;
lParam: LongInt): Bool; export;
```

DialogFunc: Platzhalter für den Namen der anwendungsunterstützten Funktion.

hDlg: Bezeichnet das Dialogfenster, das die Botschaft empfängt.

wMsg: Bestimmt die Nummer der Botschaft.

wParam: Bestimmt 16 Bits zusätzliche, von der Botschaft abhängige Information.

lParam: Bestimmt 32 Bits zusätzliche, von der Botschaft abhängige Information.

Rückgabewert Mit Ausnahme der Reaktion auf die *wm_InitDialog*-Botschaft sollte die Dialogfunktion einen Wert ungleich 0 zurückliefern, wenn die Funktion die Botschaft bearbeitet, und 0, falls sie es nicht tut. Als Antwort auf eine *wm_InitDialog*-Botschaft sollte die Dialogfunktion 0 zurückliefern, wenn sie die Funktion *SetFocus* aufruft, um den Fokus auf eines der Steuerelemente im Dialogfenster zu setzen. Andernfalls sollte sie einen Wert ungleich 0 zurückliefern. In diesem Fall wird Windows den Fokus auf das erste Steuerelement im Dialogfenster setzen, dem der Fokus gegeben werden kann.

Hinweis Die Dialogfunktion wird nur benutzt, wenn die Dialogklasse für das Dialogfenster verwendet wird. Dies ist die voreingestellte Klasse. Sie wird benutzt, wenn ausdrücklich keine andere Klasse in der Dialogfenstervorlage vorgegeben wird. Obwohl die Dialogfunktion einer Fensterfunktion sehr ähnlich ist, darf sie nicht die Funktion *DefWindowProc* aufrufen, um ungewollte Botschaften zu bearbeiten. Ungewollte Botschaften werden intern von der Fensterfunktion der Dialogklasse bearbeitet.

Die Adresse der Dialogfunktion, die als Parameter *DialogFunc* übergeben wird, muß durch die Verwendung der Funktion *MakeProcInstance* erzeugt werden.

Querverweis *DefDlgProc, MakeProcInstance, wm_InitDialog*

CreateDialogIndirect

Syntax
```
function CreateDialogIndirect(Instance: THandle;
DialogTemplate: Pointer; WndParent: HWnd; DialogFunc:
TFarProc): HWnd;
```

Beschreibung Diese Funktion erzeugt ein nichtmodales Dialogfenster, das die Größe, den Stil und die Steuerelemente besitzt, die durch die die vom Parameter *DialogTemplate* vorgegebene Dialogfenstervorlage definiert werden.

Der Parameter *WndParent* bezeichnet ein das Dialogfenster besitzendes Anwendungsfenster. Die Dialogfunktion, auf die der Parameter *DialogFunc* zeigt, bearbeitet jede Botschaft, die vom Dialogfenster empfangen wird.

CreateDialogIndirect sendet eine *wm_InitDialog*-Botschaft an die Dialogfunktion, bevor sie das Dialogfenster anzeigt. Diese Botschaft erlaubt der Dialogfunktion, die Steuerelemente des Dialogfensters zu initialisieren.

CreateDialogIndirect kehrt sofort zurück, nachdem sie das

Dialogfenster erzeugt hat, sie wartet nicht, bis das Dialogfenster die Eingabebearbeitung beginnt.

Instance: Bezeichnet eine Instanz des Moduls, dessen ausführbare Datei die Vorlage des Dialogfenster enthält.

DialogTemplate: Zeigt auf einen Speicherblock, der eine *TDlgTemplate*-Datenstruktur enthält.

WndParent: Bezeichnet das Fenster, welches das Dialogfenster besitzt.

DialogFunc: Ist die Adresse der Prozedurinstanz der Dialogfunktion. Details entnehmen Sie dem folgenden Hinweis.

Rückgabewert

Der Rückgabewert ist bei erfolgreicher Ausführung der Funktion das Fenster-Handle des Dialogfensters. Er ist 0, wenn die Funktion das Dialogfenster oder eines der Steuerelemente im Dialogfenster nicht erzeugen kann.

Hinweis

Verwenden Sie den Stil *ws_Visible* in der Vorlage des Dialogfensters, wenn das Dialogfenster nach der Erzeugung im übergeordneten Fenster erscheinen soll.

Ein Dialogfenster kann bis zu 255 Steuerelemente haben.

Callback-Funktion

```
function DialogFunc (hdlg: HWnd; wMsg, wParam: Word;
lParam: LongInt): Bool; export;
```

DialogFunc: Platzhalter für den Namen der anwendungsunterstützten Funktion.

hDlg: Bezeichnet das Dialogfenster, das die Botschaft empfängt.

wMsg: Bezeichnet die Nummer der Botschaft.

wParam: Bezeichnet 16 Bits zusätzliche, von der Botschaft abhängige Information.

lParam: Bezeichnet 32 Bits zusätzliche, von der Botschaft abhängige Information.

Rückgabewert

Mit Ausnahme der Reaktion auf die *wm_InitDialog*-Botschaft sollte die Dialogfunktion einen Wert ungleich 0 zurückliefern, wenn die Funktion die Botschaft bearbeitet, und 0, falls sie es nicht tut. Als Antwort auf eine *wm_InitDialog*-Botschaft sollte die Dialogfunktion 0 zurückliefern, wenn sie die Funktion *SetFocus* aufruft, um den Fokus auf eines der Steuerelemente im Dialogfenster zu setzen. Andernfalls sollte sie einen Wert ungleich 0 zurückliefern, und in diesem Fall wird Windows den Fokus auf das erste Steuerelement im Dialogfenster setzen, dem der Fokus gegeben werden kann.

Hinweis

Die Dialogfunktion wird nur benutzt, wenn die Dialogklasse für das Dialogfenster verwendet wird. Dies ist die voreingestell-

te Klasse und wird benutzt, wenn ausdrücklich keine andere Klasse in der Dialogfenstervorlage vorgegeben wird. Obwohl die Dialogfunktion einer Fensterfunktion sehr ähnlich ist, darf sie nicht die Funktion *DefWindowProc* aufrufen, um ungewollte Botschaften zu bearbeiten. Ungewollte Botschaften werden intern von der Fensterfunktion der Dialogklasse bearbeitet.

Die Adresse der Dialogfunktion, die als Parameter *DialogFunc* übergeben wird, muß durch die Verwendung der Funktion *MakeProcInstance* erzeugt werden.

Querverweis *DefDlgProc, MakeProcInstance, wm_InitDialog*

CreateDialogIndirectParam

Syntax
```
function CreateDialogIndirectParam(Instance: THandle;
DialogTemplate: Pointer; WndParent: HWnd; DialogFunc:
TFarProc; InitParam: LongInt): HWnd;
```

Beschreibung Diese Funktion erzeugt ein nichtmodales Dialogfenster, sendet eine *wm_InitDialog*-Botschaft an die Dialogfunktion, bevor sie das Dialogfenster anzeigt, und übergibt *InitParam* als Parameter *lParam* der Botschaft. Diese Botschaft erlaubt der Dialogfunktion, die Steuerelemente des Dialogfensters zu initialisieren. Im übrigen ist diese Funktion identisch mit der Funktion *CreateDialogIndirect*.

Weitere Informationen über das Erzeugen eines nichtmodalen Dialogfensters finden Sie in der Beschreibung der Funktion *CreateDialogIndirect*.

Instance: Bezeichnet eine Instanz des Moduls, dessen ausführbare Datei die Vorlage des Dialogfensters enthält.

DialogTemplate: Zeigt auf einen Speicherblock, der eine *TDlgTemplate* Datenstruktur enthält.

WndParent: Bezeichnet das Fenster, welches das Dialogfenster besitzt.

DialogFunc: Ist die Adresse der Prozedurinstanz der Dialogfunktion. Details entnehmen Sie der Beschreibung der Funktion *CreateDialogIndirect*.

InitParam: Ist ein 32-Bit Wert, den *CreateDialogIndirectParam* der Dialogfunktion übergibt, wenn sie das Dialogfenster erzeugt.

Rückgabewert Der Rückgabewert bei erfolgreicher Ausführung der Funktion ist das Fenster-Handle des Dialogfensters. Er ist 0, wenn die Funktion das Dialogfenster oder eines der Steuerelemente im Dialogfenster nicht erzeugen kann.

Querverweis *DefDlgProc, MakeProcInstance, wm_InitDialog*

CreateDialogParam

Syntax

```
function CreateDialogParam(Instance: THandle;
TemplateName: PChar; WndParent: HWnd; DialogFunc:
TFarProc; InitParam: LongInt): HWnd;
```

Beschreibung

Diese Funktion erzeugt ein nichtmodales Dialogfenster, das die Größe, den Stil und die Steuerelemente besitzt, die durch die vom Parameter *TemplateName* bezeichnete Dialogfenstervorlage definiert werden. Der Parameter *WndParent* bezeichnet das Anwendungsfenster, welches das Dialogfenster besitzt. Die Dialogfunktion, auf die der Parameter *DialogFunc* zeigt, bearbeitet jede Botschaft, die vom Dialogfenster empfangen wird.

CreateDialogParam sendet die Botschaft *wm_InitDialog* an die Dialogfunktion, bevor das Dialogfenster angezeigt wird. Diese Botschaft ermöglicht die Initialisierung der Steuerelemente des Dialogfensters durch die Dialogfunktion. Als Parameter *lParam* der Botschaft wird *InitParam* verwendet. *CreateDialogParam* kehrt unmittelbar nach Erzeugung des Dialogfensters zurück, der Beginn der Eingabebearbeitung wird nicht abgewartet.

Instance: Bezeichnet eine Instanz des Moduls, dessen ausführbare Datei die Vorlage des Dialogfensters enthält.

TemplateName: Zeigt auf einen nullterminierten String, der die Vorlage des Dialogfensters benennt.

WndParent: Bezeichnet das Fenster, welches das Dialogfenster besitzt.

DialogFunc: Ist die Adresse der Prozedurinstanz der Dialogfunktion. Details im folgenden »Hinweis«.

InitParam: Wird als Parameter *lParam* der Botschaft *wm_InitDialog* übergeben, wenn das Dialogfenster erzeugt wird.

Rückgabewert

Der Rückgabewert bei erfolgreicher Ausführung der Funktion ist das Fenster-Handle des Dialogfensters. Er ist 0, wenn die Funktion das Dialogfenster nicht erzeugen kann.

CreateDIBitmap

Syntax

```
function CreateDIBitmap(DC: HDC; var InfoHeader:
TBitmapInfoHeader; dwUsage: LongInt; InitBits: PChar;
var InitInfo: TBitmapInfo; wUsage: Word): HBitmap;
```

Beschreibung

Diese Funktion erzeugt ein gerätespezifisches Speicher-Bitmap aus einer geräteunabhängigen Bitmap-Beschreibung (DIB) und setzt optional Bits im Bitmap.

DC: Bezeichnet den Gerätekontext.

InfoHeader: Eine *TBitmapInfoHeader*-Struktur, die Größe und Format des geräteunabhängigen Bitmap beschreibt.

dwUsage: Zeigt an, ob das Speicher-Bitmap initialisiert werden soll. Wenn *dwUsage* auf *cbm_Init* gesetzt ist, initialisiert *CreateDIBitmap* das Bitmap mit den in *InitBits* und *InitInfo* angegebenen Bits.

InitBits: Zeigt auf ein Byte-Array, das die ursprünglichen Bitmap-Werte enthält. Das Format der Bitmap-Werte hängt vom Feld *biBitCount* der *BitmapInfoHeader*-Struktur ab. Diese ist mit der durch *InitInfo* bezeichneten Datenstruktur *TBitmapInfo* verknüpft.

InitInfo: Eine *TBitmapInfo*-Datenstruktur, die die Abmessungen und das Farbformat von *InitBits* beschreibt.

wUsage: Gibt an, ob die *bmiColors*-Elemente der *InitInfo*-Datenstruktur explizite RGB-Werte oder Indizes in die aktuell realisierte virtuelle Palette aufnehmen. Der Parameter *wUsage* muß einen der folgenden Werte annehmen:

Wert	Bedeutung
DIB_Pal_Colors	Die Farbtabelle besteht aus einem Array von 16-Bit-Indizes in die gegenwärtig realisierte virtuelle Palette.
DIB_RGB_Colors	Die Farbtabelle enthält explizite RGB-Werte.

Rückgabewert Der Rückgabewert bezeichnet ein Bitmap, wenn die Funktion erfolgreich ausgeführt wurde, andernfalls ist er 0.

CreateDIBPatternBrush

Syntax
```
function CreateDIBPatternBrush(PackedDIB: THandle;
Usage: Word): HBrush;
```

Beschreibung Diese Funktion erzeugt einen virtuellen Pinsel mit dem Muster, das von dem durch den Parameter *PackedDIB* definierten geräteunabhängigen Bitmap (DIB) angegegeben wird. Der Pinsel kann später für jede Geräteeinheit ausgewählt werden, die Rasteroperationen unterstützt. Weitere Informationen finden Sie bei der Beschreibung der Funktion *GetDeviceCaps* und in Kapitel 1 »Windows-Stile und -Konstanten« im Abschnitt »Gerätefähigkeiten«.

PackedDIB: Bezeichnet ein globales Speicherobjekt, das ein gepacktes geräteunabhängiges Bitmap enthält. Um dieses Handle zu erhalten, ruft eine Anwendung die Funktion *GlobalAlloc* auf, um einen Block globalen Speichers zu reservieren, und füllt dann den Speicher mit dem gepackten DIB. Ein

gepacktes DIB besteht aus einer *TBitmapInfo*-Datenstruktur, unmittelbar gefolgt von einem Byte-Array, das die Pixel des Bitmap definiert.

Usage: Gibt an, ob die *bmiColors*-Elemente der *TBitmapInfo*-Datenstruktur explizite RGB-Werte oder Indizes in die gegenwärtig realisierte virtuelle Palette enthalten. Der Parameter *Usage* muß einen der folgenden Werte annehmen:

Wert	Bedeutung
DIB_Pal_Colors	Die Farbtabelle besteht aus einem Array von 16-Bit-Indizes in die gegenwärtig realisierte virtuelle Palette.
DIB_RGB_Colors	Die Farbtabelle enthält explizite RGB-Werte.

Rückgabewert Der Rückgabewert bezeichnet einen virtuellen Pinsel, wenn die Funktion erfolgreich ausgeführt wurde, andernfalls ist er 0.

Querverweis *GetDeviceCaps*

CreateDiscardableBitmap

Syntax
```
function CreateDiscardableBitmap(DC: HDC; Width,
Height: Integer): HBitmap;
```

Beschreibung Diese Funktion erzeugt ein verwerfbares Bitmap, das mit dem durch den Parameter *DC* bezeichneten Gerätekontext kompatibel ist. Das Bitmap hat die gleiche Anzahl an Farbebenen bzw. das gleiche Bits-pro-Pixel-Format wie die angegebene Geräteeinheit.

Eine Anwendung kann dieses Bitmap als aktuelles Bitmap für ein Speichergerät auswählen, das mit dem vom Parameter *DC* angegebenen kompatibel ist.

DC: Bezeichnet einen Gerätekontext.

Width: Bestimmt die Breite des Bitmap in Bits.

Height: Bestimmt die Höhe des Bitmap in Bits.

Rückgabewert Der Rückgabewert bezeichnet ein Bitmap, wenn die Funktion erfolgreich ausgeführt wurde, andernfalls ist er 0.

Hinweis Windows kann ein Bitmap, das von dieser Funktion erzeugt wurde, nur verwerfen, wenn eine Anwendung es nicht für einen Bildschirmkontext ausgewählt hat. Verwirft Windows das Bitmap, solange es nicht ausgewählt ist und versucht die Anwendung später, es auszuwählen, wird die Funktion *SelectObject* 0 zurückliefern. In diesem Fall sollte die Anwendung das Handle des Bitmap durch Verwendung von *DeleteObject* entfernen.

CreateEllipticRgn

Syntax
```
function CreateEllipticRgn(X1, Y1, X2, Y2: Integer):
HRgn;
```

Beschreibung Diese Funktion erzeugt eine elliptische Region.

X1: Bestimmt die x-Koordinate der oberen linken Ecke des umgebenden Rechtecks der Ellipse.

Y1: Bestimmt die y-Koordinate der oberen linken Ecke des umgebenden Rechtecks der Ellipse.

X2: Bestimmt die x-Koordinate der unteren rechten Ecke des umgebenden Rechtecks der Ellipse.

Y2: Bestimmt die y-Koordinate der unteren rechten Ecke des umgebenden Rechtecks der Ellipse.

Rückgabewert Der Rückgabewert bezeichnet eine neue Region, wenn die Funktion erfolgreich ausgeführt wurde, andernfalls ist er 0.

Hinweis Die Breite des Rechtecks, die durch den absoluten Wert der Differenz *X2 - X1* gegeben ist, darf 32.767 Einheiten nicht überschreiten.

Diese Beschränkung gilt auch für die Höhe des Rechtecks (*Y2 - Y1*).

CreateEllipticRgnIndirect

Syntax
```
function CreateEllipticRgnIndirect(var Rect: TRect):
HRgn;
```

Beschreibung Diese Funktion erzeugt eine elliptische Region.

Rect: Eine *TRect*-Datenstruktur, die die Koordinaten der oberen linken und der unteren rechten Ecke des Begrenzungsrechtecks der Ellipse enthält.

Rückgabewert Der Rückgabewert bezeichnet eine neue Region, wenn die Funktion erfolgreich ausgeführt wurde, andernfalls ist er 0.

Hinweis Die Breite des Rechtecks darf 32.767 Einheiten nicht überschreiten, diese Beschränkung gilt auch für die Höhe des Rechtecks.

CreateFont

Syntax
```
function CreateFont(Height, Width, Escapement,
Orientation, Weight: Integer; Italic, Underline,
StrikeOut, CharSet, OutputPrecision, ClipPrecision,
Quality, PitchAndFamily: Byte; Facename: PChar): HFont;
```

Beschreibung Diese Funktion erzeugt eine virtuelle Schrift, die die angegebenen Merkmale hat. Die virtuelle Schrift kann später für jede Geräteeinheit als aktuelle Schrift ausgewählt werden.

Height: Gibt die gewünschte Höhe der Schrift in virtuellen Einheiten an. Die Schrifthöhe kann auf drei Arten angegeben werden:

- Wenn *Height* größer als 0 ist, wird der Wert in Geräteeinheiten umgewandelt an die Zellenhöhe der verfügbaren Schriften angepaßt.
- Wenn *Height* 0 ist, wird eine angemessene voreingestellte Größe verwendet.
- Wenn *Height* kleiner als 0 ist, wird er in Geräteeinheiten umgewandelt, und der absolute Wert wird an die Zellenhöhe der verfügbaren Schriften angepaßt.

Für alle Höhenvergleiche sucht der Schriftabbilder nach der höchsten Schrift, die die gewünschte Größe nicht überschreitet und – falls es eine solche Schrift nicht gibt – nach der kleinsten verfügbaren Schrift.

Width: Bestimmt die durchschnittliche Breite von Zeichen in der Schrift in virtuellen Einheiten. Wenn *Width* 0 ist, wird das Höhen-/Seitenverhältnis der Geräteeinheit mit dem Höhen-/Seitenverhältnis der verfügbaren Schriften verglichen, um das passendste Gegenstück, das durch den absoluten Wert der Differenz ermittelt wird, zu suchen.

Escapement: Bestimmt den Winkel jeder Textzeile, die in der Schrift geschrieben wird, in Zehntelgraden relativ zum unteren Seitenrand.

Orientation: Bestimmt den Winkel der unteren Grenze jedes Zeichens, in zehntel Grad relativ zum unteren Seitenrand.

Weight: Bestimmt die gewünschte Schriftstärke im Bereich von 0 bis 1000 (400 ist normal, 700 ist fett). Wenn *Weight* 0 ist, wird ein voreingestelltes Maß verwendet. Außerdem kann auf die in Kapitel 1 »Windows-Stile und -Konstanten« im Abschnitt »fw_Xxxx« beschriebenen Werte zurückgegriffen werden.

Italic: Bestimmt, ob die Schrift kursiv ist.

Underline: Bestimmt, ob die Schrift unterstrichen ist.

StrikeOut: Gibt an, ob die Zeichen der Schrift durchgestrichen sind.

CharSet: Gibt den gewünschten Zeichensatz an. Die folgenden Werte sind vordefiniert: *ANSI_CharSet, OEM_CharSet* sowie *SYMBOL_CharSet.*

Der OEM-Zeichensatz ist systemabhängig, Schriften mit anderen Zeichensätzen können im System existieren. Wenn eine Anwendung eine Schrift mit einem unbekannten Zeichensatz verwendet, sollte sie nicht versuchen, Strings zu übersetzen oder auszuwerten, die in dieser Schrift dargestellt werden sollen. Statt dessen sollten die Strings direkt an den Ausgabegerätetreiber übergeben werden.

OutputPrecision: Gibt die gewünschte Ausgabepräzision an. Die Ausgabepräzision bestimmt, wie genau die gewünschte Höhe, Breite, Zeichenausrichtung, Flucht und Neigung der Schrift bei der Ausgabe dargestellt wird. Er kann jeden der folgenden Werte annehmen: *Out_Character_Precis*, *Out_Default_Precis*, *Out_String_Precis* oder *Out_Stroke_Precis*.

ClipPrecision: Gibt die gewünschte Clipping-Präzision an, die bestimmt, wie Zeichen abgeschnitten werden sollen, die teilweise außerhalb des Clipping-Bereiches liegen. Er kann jeden der folgenden Werte annehmen: *Clip_Character_Precis*, *Clip_-Default_Precis* oder *Clip_Stroke_Precis*.

Quality: Gib die gewünschte Ausgabequalität an, die bestimmt, wie sorgfältig GDI versucht, die virtuellen Schriftattribute denen einer tatsächlichen physikalischen Schrift anzugleichen. Er kann jeden der folgenden Werte annehmen: *Default_Quality*, *Draft_Quality* oder *Proof_Quality* (siehe Kapitel 1 »Windows-Stile und -Konstanten«).

PitchAndFamily: Gibt Zeichenabstand und Herkunft der Schrift an. Die zwei niederwertigen Bits bestimmen den Zeichenabstand der Schrift und können jeden der folgenden Werte annehmen: *Default_Pitch*, *Fixed_Pitch* oder *Variable_Pitch*.

Die vier höherwertigen Bits des Feldes geben die Schriftenfamilie an und können jeden der folgenden Werte annehmen: *ff_Decorative*, *ff_DontCare*, *ff_Modern*, *ff_Roman*, *ff_Script* oder *ff_Swiss*. Diese Konstanten sind in Kapitel 1 »Windows-Stile und -Konstanten« im Abschnitt »ff_Xxxx« beschrieben.

Facename: Zeigt auf einen mit 0 abgeschlossenen String, der den Typennamen der Schrift angibt. Die Länge dieses Strings darf 30 Zeichen nicht überschreiten. Die Funktion *EnumFonts* kann verwendet werden, um die Typennamen aller aktuell verfügbaren Schriften aufzuzählen.

Rückgabewert Der Rückgabewert bezeichnet eine virtuelle Schrift, wenn die Funktion erfolgreich ausgeführt wurde, andernfalls ist er 0.

Hinweis *CreateFont* erzeugt keine neue Schrift, die Funktion wählt nur das passendste Gegenstück aus den Schriften aus, die in der physikalischen Schriftensammlung der GDI verfügbar sind.

Querverweis *EnumFonts*

CreateFontIndirect

Syntax
```
function CreateFontIndirect(var LogFont: TLogfont): HFont;
```

Beschreibung Diese Funktion erzeugt eine virtuelle Schrift mit den Merkmalen, die von der Datenstruktur, auf die der Parameter *LogFont* zeigt, vorgegeben werden. Die Schrift kann später von jeder Geräteeinheit als aktuelle Schrift ausgewählt werden.

LogFont: Eine *TLogFont*-Datenstruktur, die die Merkmale der virtuellen Schrift bestimmt.

Rückgabewert Der Rückgabewert bezeichnet eine virtuelle Schrift, wenn die Funktion erfolgreich ausgeführt wurde, andernfalls ist er 0.

Hinweis *CreateFontIndirect* erzeugt eine virtuelle Schrift, die alle angegebenen Merkmale besitzt. Wenn die Schrift mit der Funktion *SelectObject* ausgewählt wird, versucht der Schriftabbilder der GDI, eine zur virtuellen Schrift passende physikalische Schrift zu finden. Findet er keine exakt passende Schrift, stellt er eine Alternative zur Verfügung, die so viele der gewünschten Merkmale wie möglich aufweist.

CreateHatchBrush

Syntax
```
function CreateHatchBrush(Index: Integer; Color: TColorRef): HBrush;
```

Beschreibung Diese Funktion erzeugt einen virtuellen Pinsel mit der angegebenen Schraffur und Farbe. Der Pinsel kann anschließend von jeder Geräteeinheit als aktueller Pinsel ausgewählt werden.

Index: Bestimmt den Schraffurstil des Pinsels. Er kann jeden der folgenden Werte annehmen:

Wert	Bedeutung
hs_BDiagonal	45-Grad abwärts schraffiert (von links nach rechts) (\\\\\).
hs_Cross	Horizontal und vertikal überkreuz schraffiert (+++++).
hs_*DiagCross*	45-Grad überkreuz schraffiert (xxxxx).
hs_FDiagonal	45-Grad aufwärts schraffiert (von links nach rechts) (/////).
hs_Horizontal	Horizontal schraffiert (-----).
hs_Vertical	Vertikal schraffiert (|||||).

Color: Bestimmt die Vordergrundfarbe des Pinsels bzw. die Farbe der Schraffur.

Rückgabewert Der Rückgabewert bezeichnet einen virtuellen Pinsel, wenn die Funktion erfolgreich ausgeführt wurde, andernfalls ist er 0.

CreateIC

Syntax
```
function CreateIC(DriverName, DeviceName, Output:
PChar; InitData: Pointer): HDC;
```

Beschreibung Diese Funktion erzeugt einen Informationskontext für die angegebene Geräteeinheit, der es ermöglicht, schnell Informationen über die Geräteeinheit zu erlangen, ohne einen Gerätekontext zu erzeugen.

DriverName: Zeigt auf einen nullterminierten String, der den DOS-Dateinamen (ohne Namenserweiterung) des Gerätetreibers angibt (zum Beispiel Epson).

DeviceName: Zeigt auf einen nullterminierten String, der den Namen der Geräteeinheit angibt (z.B. »Epson FX-80«). Der Parameter *DeviceName* wird verwendet, wenn das Modul mehr als eine Geräteeinheit unterstützt.

Output: Zeigt auf einen nullterminierten String, der den DOS-Dateinamen oder den Gerätenamen für das physikalische Ausgabemedium angibt (Datei oder Ausgabeanschluß).

InitData: Zeigt auf gerätespezifische Initialisierungsdaten für den Gerätetreiber. Der Parameter *InitData* muß 0 sein, wenn der Gerätetreiber die vom Anwender durch die Systemsteuerung angegebene voreingestellte Initialisierung (falls vorhanden) verwenden soll.

Rückgabewert Der Rückgabewert bezeichnet einen Informationskontext für die angegebene Geräteeinheit, wenn die Funktion erfolgreich ausgeführt wurde, andernfalls ist er 0.

Hinweis DOS-Gerätenamen folgen den DOS-Konventionen, ein Doppelpunkt (:) am Ende ist empfehlenswert, jedoch optional. Windows entfernt den abschließenden Doppelpunkt, so daß ein Gerätename, der mit einem Doppelpunkt endet, dem gleichen Anschluß zugeordnet wird, wie der gleiche Name ohne einen Doppelpunkt. Die Treiber- und Anschlußnamen dürfen keine führenden oder abschließenden Leerzeichen enthalten. GDI-Ausgabefunktionen können nicht mit Informationskontexten benutzt werden.

CreateIcon

Syntax
```
function CreateIcon(Instance: THandle; Width, Height:
Integer; Planes, BitsPixel: Byte; ANDbits, XORbits:
Pointer): HIcon;
```

Beschreibung Diese Funktion erzeugt ein Symbol, das die angegebene Breite, Höhe und Bitmuster hat.

Instance: Bezeichnet eine Instanz des Moduls, das das Symbol erzeugt hat.

Width: Bestimmt die Breite des Symbols in Pixeln.

Height: Bestimmt die Höhe des Symbols in Pixeln.

Planes: Bestimmt die Anzahl der Ebenen in der **xor**-Maske des Symbols.

BitsPixel: Bestimmt die Anzahl der Bits pro Pixel in der **xor**-Maske des Symbols.

ANDbits: Zeigt auf ein Byte-Array, das die Bitwerte für die **and**-Maske des Symbols enthält. Dieses Array muß eine monochrome Maske bezeichnen.

XORbits: Zeigt auf ein Byte-Array, das die Bitwerte für die **xor**-Maske des Symbols enthält. Dies können die Bits eines monochromen oder geräteabhängigen farbigen Bitmaps sein.

Rückgabewert Der Rückgabewert bezeichnet ein Symbol, wenn die Funktion erfolgreich ausgeführt wurde, andernfalls ist er 0.

CreateMenu

Syntax `function CreateMenu: HMenu;`

Beschreibung Diese Funktion erzeugt ein Menü. Das Menü ist anfangs leer, kann aber mit Menüoptionen gefüllt werden, indem man die Funktionen *AppendMenu* oder *InsertMenu* verwendet.
Diese Funktion hat keine Parameter.

Rückgabewert Der Rückgabewert bezeichnet bei erfolgreicher Ausführung der Funktion ein neu erzeugtes Menü. Er ist 0, wenn das Menü nicht erzeugt werden kann.

Querverweis *AppendMenu, InsertMenu*

CreateMetaFile

Syntax `function CreateMetaFile(Filename: PChar): THandle;`

Beschreibung Diese Funktion erzeugt einen Metafile-Gerätekontext.

FileName: Zeigt auf einen nullterminierten String, der den Namen des Metafile angibt. Wenn der Parameter *FileName* 0 ist, wird ein Gerätekontext für ein Speicher-Metafile zurückgeliefert.

Rückgabewert Der Rückgabewert bezeichnet einen Metafile-Gerätekontext, wenn die Funktion erfolgreich ausgeführt wurde, andernfalls ist er 0.

CreatePalette

Syntax
```
function CreatePalette(var LogPalette: TLogPalette): HPalette;
```

Beschreibung Diese Funktion erzeugt eine virtuelle Farbpalette.

LogPalette: Eine *TLogPalette*-Datenstruktur, die Informationen über die Farben der virtuellen Palette enthält.

Rückgabewert Der Rückgabewert bezeichnet eine virtuelle Palette, wenn die Funktion erfolgreich ausgeführt wurde, andernfalls ist er 0.

CreatePatternBrush

Syntax
```
function CreatePatternBrush(Bitmap: HBitmap): HBrush;
```

Beschreibung Diese Funktion erzeugt einen virtuellen Pinsel, der das vom Parameter *Bitmap* angegebene Muster hat. Der Pinsel kann später für jede Geräteeinheit ausgewählt werden, die Rasteroperationen unterstützt. Weitere Informationen finden Sie bei der Beschreibung der Funktion *GetDeviceCaps* und in Kapitel 1 »Windows-Stile und -Konstanten« im Abschnitt »dc_Xxxx«.

Bitmap: Bezeichnet das Bitmap, vorausgesetzt, daß es durch eine der Funktionen *CreateBitmap*, *CreateBitmapIndirect*, *LoadBitmap* oder *CreateCompatibleBitmap* erzeugt worden ist. Die minimale Größe eines Bitmaps, das in einem Füllmuster benutzt werden soll, beträgt 8-auf-8.

Rückgabewert Der Rückgabewert bezeichnet einen virtuellen Pinsel, wenn die Funktion erfolgreich ausgeführt wurde, andernfalls ist er 0.

Hinweis Ein Musterpinsel kann gelöscht werden, ohne das zugeordnete Bitmap zu beeinflussen, indem man die Funktion *DeleteObject* verwendet. Damit kann das Bitmap benutzt werden, um eine beliebige Anzahl von Musterpinseln zu erzeugen.

Ein Pinsel, der unter Verwendung eines monochromen (eine Ebene, ein Bit pro Pixel) Bitmap erzeugt wurde, wird mit der aktuellen Text- und Hintergrundfarbe gezeichnet. Pixel, die durch ein auf 0 gesetztes Bit repräsentiert werden, werden in der aktuellen Textfarbe gezeichnet. Pixel, bei denen das Bit auf 1 gesetzt ist, werden in der aktuellen Hintergrundfarbe gezeichnet.

Querverweis *CreateBitmap*, *CreateBitmapIndirect*, *LoadBitmap*, *CreateCompatibleBitmap*, *DeleteObject*, *GetDeviceCaps*

CreatePen

Syntax
```
function CreatePen(PenStyle, Width: Integer; Color:
TColorRef): HPen;
```

Beschreibung Diese Funktion erzeugt einen virtuellen Stift, der den Stil, die Breite und Farbe hat, die angegeben werden. Der Stift kann später als aktueller Stift für jede Geräteeinheit ausgewählt werden.

PenStyle: Gibt den Stil des Stifts an. Er kann jeden der folgenden Werte annehmen (vergleichen Sie hierzu Kapitel 1 »Windows-Stile und -Konstanten«, Abschnitt »ps_Xxxx«):

Stiftstil	Wert
ps_Solid	0
ps_Dash	1
ps_Dot	2
ps_DashDot	3
ps_DashDotDot	4
ps_Null	5
ps_InsideFrame	6

ps_InsideFrame: Wenn die Breite des Stifts größer als 1 und der Stiftstil *ps_InsideFrame* ist, wird die Linie innerhalb des Rahmens aller Grundfunktionen mit Ausnahme von Polygonen und Linienzügen gezeichnet. Der Stift wird mit einer virtuellen (angenäherten) Farbe gezeichnet, wenn die Stiftfarbe keinem verfügbaren RGB-Wert entspricht. Der Stil *ps_InsideFrame* ist identisch mit *ps_Solid*, wenn die Stiftbreite kleiner oder gleich 1 ist.

Width: Bestimmt die Breite des Stifts in virtuellen Einheiten.

Color: Bestimmt die Farbe des Stifts.

Rückgabewert Der Rückgabewert bezeichnet einen virtuellen Stift, wenn die Funktion erfolgreich ausgeführt wurde, andernfalls ist er 0.

Hinweis Stifte mit einer physikalischen Breite von mehr als einem Pixel haben immer den Stil *ps_Null* oder *ps_Solid* oder werden angenähert, wenn der Stiftstil *ps_InsideFrame* ist.

CreatePenIndirect

Syntax
```
function CreatePenIndirect(var LogPen: TLogPen): HPen;
```

Beschreibung Diese Funktion erzeugt einen virtuellen Stift, der den Stil, die Breite und Farbe hat, die von der Datenstruktur vorgegeben werden, auf die der Parameter *LogPen* zeigt.

LogPen: TLogPen-Datenstruktur, die Informationen über den virtuellen Stift enthält.

Rückgabewert Der Rückgabewert bezeichnet einen virtuellen Stift, wenn die Funktion erfolgreich ausgeführt wurde, andernfalls ist er 0.

Hinweis Stifte mit einer physikalischen Breite von mehr als einem Pixel haben immer den Stil *ps_Null* oder *ps_Solid* oder werden angenähert, wenn der Stiftstil *ps_InsideFrame* ist.

CreatePolygonRgn

Syntax
```
function CreatePolygonRgn(var Points; Count,
PolyFillMode: Integer): HRgn;
```

Beschreibung Diese Funktion erzeugt eine Polygon-Region.

Points: Zeigt auf ein Array von *TPoint*-Datenstrukturen. Jeder Punkt gibt die x- und y-Koordinaten eines Scheitelpunktes des Polygons an.

Count: Gibt die Anzahl der Punkte im Array an.

PolyFillMode: Bestimmt den Polygon-Füllmodus, der benutzt werden soll, um die Region auszufüllen. Er kann *Alternate* oder *Winding* sein. Die Modi werden in Kapitel 1 »Windows-Stile und -Konstanten« im Abschnitt »Polygon-Füllmodi« beschrieben.

Rückgabewert Der Rückgabewert bezeichnet eine neue Region, wenn die Funktion erfolgreich ausgeführt wurde, andernfalls ist er 0.

Querverweis *SetPolyFillMode*

CreatePolyPolygonRgn

Syntax
```
function CreatePolyPolygonRgn(var Points; var
PolyCounts; Count, PolyFillMode: Integer): HRgn;
```

Beschreibung Diese Funktion erzeugt eine Region, die aus einer Reihe von geschlossenen Polygonen besteht. Die Region wird unter Verwendung des durch den Parameter *PolyFillMode* angegebenen Modus ausgefüllt. Die Polygone können sich überschneiden, müssen es aber nicht.

Points: Zeigt auf ein Array von *TPoint*-Datenstrukturen, die die Scheitelpunkte der Polygone bestimmen. Jedes Polygon muß geschlossen sein, dies wird nicht automatisch durchgeführt. Die Polygone werden nacheinander angegeben.

PolyCounts: Zeigt auf ein Integer-Array. Jeder Integer bestimmt die Anzahl von Punkten eines der Polygone des *Points*-Arrays.

Count: Gibt die Gesamtanzahl von Integerzahlen im Array *PolyCounts* an.

PolyFillMode: Gibt den Füllmodus für die Region an. Der Parameter kann einen der folgenden Werte annehmen:

Wert	Bedeutung
Alternate	Wählt Alternate-Modus.
Winding	Wählt Winding-Modus.

Rückgabewert Der Rückgabewert bezeichnet die Region, wenn die Funktion erfolgreich ausgeführt wurde, andernfalls ist er 0.

Hinweis Generell unterscheiden sich die Polygon-Füllmodi nur in Fällen, wo ein komplexes, überlappendes Polygon gefüllt werden muß (zum Beispiel ein fünfseitiges Polygon, das einen fünfzackigen Stern mit einem Fünfeck in der Mitte bildet). In solchen Fällen füllt der *Alternate*-Modus jede andere eingeschlossene Region innerhalb des Polygons (d.h. die Punkte des Sterns), der *Winding*-Modus jedoch füllt alle Regionen (d.h. die Punkte und das Fünfeck).

Wenn der Füllmodus *Alternate* ist, füllt die GDI den Bereich zwischen geraden und ungeraden Polygonseiten auf jeder Scan-Zeile. Das heißt, daß die GDI den Bereich zwischen der ersten und zweiten, der dritten und vierten Seite füllt, usw.

Um alle Teile einer Region zu füllen, veranlaßt der *Winding*-Modus die GDI, eine Umrandung zu berechnen und zu zeichnen, die die Region einschließt, aber nicht überlappt. Zum Beispiel wird im *Winding*-Modus das fünfseitige Polygon, das den Stern bildet, als ein zehnseitiges Polygon ohne sich überschneidende Seiten berechnet; der daraus resultierende Stern wird ausgefüllt.

CreatePopupMenu

Syntax
```
function CreatePopupMenu: HMenu;
```

Beschreibung Diese Funktion erzeugt das Handle eines leeren Pop-up-Menüs und gibt es zurück. Eine Anwendung fügt Einträge zu einem Pop-up-Menü hinzu, indem sie die Funktionen *InsertMenu* und *AppendMenu* aufruft. Eine Anwendung kann das Pop-up-Menü zu einem bestehenden Menü oder Pop-up-Menü hinzufügen, oder sie kann eine Auswahl im Pop-up-Menü anzeigen und verfolgen, indem sie die Funktion *TrackPopupMenu* aufruft.

Diese Funktion hat keine Parameter.

Rückgabewert Der Rückgabewert bezeichnet das neuerzeugte Menü. Er ist 0, wenn das Menü nicht erzeugt werden kann.

Querverweis *AppendMenu, InsertMenu, TrackPopupMenu*

CreateRectRgn

Syntax
```
function CreateRectRgn(X1, Y1, X2, Y2: Integer): HRgn;
```

Beschreibung Diese Funktion erzeugt eine rechteckige Region.

X1: Bestimmt die x-Koordinate der oberen linken Ecke der Region.

Y1: Bestimmt die y-Koordinate der oberen linken Ecke der Region.

X2: Bestimmt die x-Koordinate der unteren rechten Ecke der Region.

Y2: Bestimmt die y-Koordinate der unteren rechten Ecke der Region.

Rückgabewert Der Rückgabewert bezeichnet eine neue Region, wenn die Funktion erfolgreich ausgeführt werden konnte, andernfalls ist er 0.

Hinweis Die Breite des Rechtecks, die durch den absoluten Wert der Differenz *X2 - X1* gegeben ist, darf 32767 Einheiten nicht überschreiten. Diese Beschränkung gilt auch für die Höhe des Rechtecks (*Y2 - Y1*).

CreateRectRgnIndirect

Syntax
```
function CreateRectRgnIndirect(var Rect: TRect): HRgn;
```

Beschreibung Diese Funktion erzeugt eine rechteckige Region.

Rect: Eine *TRect*-Datenstruktur, die die Koordinaten der oberen linken und unteren rechten Ecke der Region enthält.

Rückgabewert Der Rückgabewert bezeichnet eine neue Region, wenn die Funktion erfolgreich ausgeführt wurde, andernfalls ist er 0.

Hinweis Die Breite des Rechtecks darf 32767 Einheiten nicht überschreiten. Diese Beschränkung gilt auch für die Höhe des Rechtecks.

CreateRoundRectRgn

Syntax
```
function CreateRoundRectRgn(X1, Y1, X2, Y2, X3, Y3:
Integer): HRgn;
```

Beschreibung Diese Funktion erzeugt eine rechteckige Region mit abgerundeten Ecken.

X1: Bestimmt die x-Koordinate der oberen linken Ecke der Region.

Y1: Bestimmt die y-Koordinate der oberen linken Ecke der Region.

X2: Bestimmt die x-Koordinate der unteren rechten Ecke der Region.

Y2: Bestimmt die y-Koordinate der unteren rechten Ecke der Region.

X3: Bestimmt die Breite der Ellipse, die verwendet wird, um die abgerundeten Ecken zu erzeugen.

Y3: Bestimmt die Höhe der Ellipse, die verwendet wird, um die abgerundeten Ecken zu erzeugen.

Rückgabewert Der Rückgabewert bezeichnet eine neue Region, wenn die Funktion erfolgreich ausgeführt wurde, andernfalls ist er 0.

Hinweis Die Breite des Rechtecks, die durch den absoluten Wert der Differenz *X2 - X1* gegeben ist, darf 32767 Einheiten nicht überschreiten. Diese Beschränkung gilt auch für die Höhe des Rechtecks (*Y2 - Y1*).

CreateSolidBrush

Syntax
```
function CreateSolidBrush(Color: TColorRef): HBrush;
```

Beschreibung Diese Funktion erzeugt einen virtuellen Pinsel, der die angegebene Farbe hat. Der Pinsel kann später für jede Geräteeinheit als aktueller Pinsel gewählt werden.

Color: TColorRef-Struktur, die die Farbe des Pinsels bestimmt.

Rückgabewert Der Rückgabewert bezeichnet einen virtuellen Pinsel, wenn die Funktion erfolgreich ausgeführt wurde, andernfalls ist er 0.

CreateWindow

Syntax
```
function CreateWindow(ClassName, WindowName: PChar;
Style: LongInt; X, Y, Width, Height: Integer;
WndParent: HWnd; Menu: HMenu; Instance: THandle; Param:
Pointer): HWnd;
```

Beschreibung Diese Funktion erzeugt ein überlappendes, ein Pop-up-, oder ein untergeordnetes Fenster. Die Funktion *CreateWindow* gibt die Fensterklasse, die Fensterüberschrift, den Fensterstil und (optional) die anfängliche Position und Größe des Fensters an. *CreateWindow* bestimmt auch das übergeordnete Fenster (falls vorhanden) und das Menü des Fenster.

Für überlappende, Pop-up- und untergeordnete Fenster übergibt *CreateWindow* die Botschaften *wm_Create, wm_GetMinMaxInfo* und *wm_NCCreate* an das Fenster. Der Parameter

lParam der Botschaft *wm_Create* enthält einen Zeiger auf eine *TCreateStruct*-Datenstruktur. Wenn der Stil *ws_Visible* gegeben ist, sendet *CreateWindow* alle Botschaften an das Fenster, die erforderlich sind, um das Fenster zu aktivieren und zu zeigen.

Bestimmt der Fensterstil eine Titelleiste, wird die Fensterüberschrift, auf die der Parameter *WindowName* zeigt, in der Titelleiste angezeigt. Benutzt man *CreateWindow*, um Steuerelemente wie Aktionsschalter, Auswahlfelder und Textfelder zu erzeugen, gibt der Parameter *WindowName* den Text der Steuerelemente an.

ClassName: Zeigt auf einen nullterminierten String, der die Fensterklasse benennt. Der Klassenname kann jeder Name sein, der durch die Funktion *RegisterClass* eingetragen wird, oder der Name jeder der vordefinierten Steuerelementeklassen, die im folgenden »Hinweis« beschrieben werden.

WindowName: Zeigt auf einen mit 0 abgeschlossenen String, der den Fensternamen darstellt.

Style: Gibt den Stil des Fensters an, das erzeugt werden soll. Es kann jede Kombination der Stile verwendet werden, die in Kapitel 1 »Windows-Stile und -Konstanten« unter den Abschnitten »ds_Xxxx«, »ws_Xxxx«, »bs_Xxxx«, »cbs_Xxxx«, »es_Xxxx«, »lbs_Xxxx«, »sbs_Xxxx« und »ss_Xxxx« beschrieben sind.

X: Bestimmt die anfängliche x-Position des Fensters. Bei einem überlappenden oder Pop-up-Fenster ist der Parameter *X* die ursprüngliche x-Koordinate der oberen linken Ecke des Fensters in Bildschirmkoordinaten. Wenn dieser Wert *cw_UseDefault* ist, wählt Windows die voreingestellte Position für die obere linke Ecke des Fensters. Bei einem untergeordneten Fenster ist *X* die x-Koordinate der oberen linken Ecke des Fensters im Client-Bereich seines übergeordneten Fensters.

Y: Gibt die anfängliche y-Position des Fensters an. Bei einem überlappenden Fenster ist *Y* die ursprüngliche y-Koordinate der oberen linken Ecke des Fensters. Bei einem Pop-up Fenster ist *Y* die y-Koordinate der oberen linken Ecke des Pop-up Fensters in Bildschirmkoordinaten. Bei Listen ist *Y* die y-Koordinate der oberen linken Ecke des Client-Bereichs des Steuerelements. Bei einem untergeordneten Fenster ist *Y* die y-Koordinate der oberen linken Ecke des untergeordneten Fensters. Alle diese Koordinaten beziehen sich auf das Fenster selbst, nicht auf den Client-Bereich des Fensters.

Width: Gibt die Breite des Fensters in Geräteeinheiten an. Bei überlappenden Fenstern ist der Parameter *Width* entweder die Breite des Fensters in Bildschirmkoordinaten oder *cw_UseDefault*. Wenn *Width cw_UseDefault* ist, wählt Windows eine

voreingestellte Breite und Höhe für das Fenster. Die voreingestellte Breite erstreckt sich von der anfänglichen x-Position bis zum rechten Rand des Bildschirms und die voreingestellte Höhe erstreckt sich von der anfänglichen y-Position zum oberen Ende des Symbolbereichs.

Height: Gibt die Höhe des Fensters in Geräteeinheiten an. Bei überlappenden Fenstern ist der Parameter *Height* die Höhe des Fensters in Bildschirmkoordinaten. Wenn der Parameter *Width cw_UseDefault* ist, ignoriert Windows *nHeight*.

WndParent: Bezeichnet das übergeordnete oder das besitzende Fenster des Fensters, das erzeugt wurde. Wenn man ein untergeordnetes oder ein zugehöriges Fenster erzeugt, muß ein gültiges Fenster-Handle angegeben werden. Ein zugehöriges Fenster ist ein überlapptes Fenster, das entfernt wird, wenn sein besitzendes Fenster entfernt, unsichtbar gemacht oder wenn sein Besitzer als Symbol dargestellt wird und das immer auf der Oberfläche seines besitzenden Fensters angezeigt wird. Bei Pop-up-Fenstern kann ein Handle angegeben werden, dies ist aber nicht erforderlich. Wenn das Fenster kein übergeordnetes Fenster hat oder es keinem anderen Fenster zugehörig ist, muß der Parameter *WndParent* auf 0 gesetzt werden.

Menu: Bezeichnet ein Menü oder einen Bezeichner eines untergeordneten Fensters. Die Bedeutung hängt vom Fensterstil ab. Bei überlappenden oder Pop-up-Fenstern bezeichnet der Parameter *Menu* das Menü, das mit dem Fenster benutzt werden soll. Er kann 0 sein, wenn das Klassenmenü verwendet werden soll. Bei untergeordneten Fenstern gibt *Menu* den Bezeichner eines untergeordneten Fensters an: einen Integer, der von einem Dialogfenster benutzt wird, um seinem übergeordneten Fenster Ereignisse mitzuteilen (wie beispielsweise die Botschaft *en_HSscroll*). Der Bezeichner des untergeordneten Fensters wird von der Anwendung ermittelt und sollte für alle untergeordneten Fenster mit dem gleichen übergeordneten Fenster eindeutig sein.

Instance: Bezeichnet die Instanz des Moduls, das dem Fenster zugeordnet werden soll.

Param: Zeigt auf einen Wert, der dem Fenster durch die *TCreate Struct*-Datenstruktur übergeben wird, auf die der Parameter *lParam* der Botschaft *wm_Create* zeigt. Wenn eine Anwendung *CreateWindow* aufruft, um ein Client-Fenster zu erzeugen, muß *lParam* auf eine *TClientCreateStruct*-Datenstruktur zeigen.

Rückgabewert Der Rückgabewert bezeichnet das neue Fenster. Er ist 0, wenn das Fenster nicht erzeugt wurde.

Hinweis Bei überlappten Fenstern, für die der Parameter *X* auf *cw_Use-Default* gesetzt ist, kann der Parameter *Y* einer der bei der

Funktion *ShowWindow* beschriebenen Parameter für Darstellungsstile sein oder er kann für das erste überlappte Fenster, das von der Anwendung erzeugt werden soll, der der Funktion *WinMain* übergebene Parameter *nCmdShow* sein.

Die folgende Übersicht beschreibt die Klassen der Steuerelemente, die von Windows vordefiniert sind.

Klasse	Bedeutung
Button	Bestimmt ein kleines rechteckiges untergeordnetes Fenster, das einen Aktionsschalter darstellt, den der Benutzer durch Anklicken ein- oder ausschalten kann. Schalter können alleine oder in Gruppen verwendet werden und können entweder beschriftet sein oder ohne Text erscheinen. Schalter ändern üblicherweise ihr Aussehen, wenn der Benutzer sie anklickt.
ComboBox	Bestimmt ein Steuerelement, das aus einem Auswahlfeld mit einem Editierfeld und einer Liste besteht. Die Liste kann zu jeder Zeit oder dann angezeigt werden, wenn der Benutzer ein Feld neben dem Auswahlfeld anwählt bzw. anklickt. Abhängig vom Stil des Kombinationfensters kann der Benutzer den Inhalt des Auswahlfeldes editieren oder nicht. Wenn die Liste sichtbar ist, verursacht das Eingeben von Zeichen im Auswahlfenster, daß der erste Listeneintrag, der zu den eingegebenen Zeichen paßt, hervorgehoben dargestellt wird. Umgekehrt bewirkt das Anwählen eines Eintrags in der Liste, daß der ausgewählte Text im Auswahlfeld angezeigt wird.
Edit	Bestimmt ein rechteckiges untergeordnetes Fenster, in dem der Benutzer mit der Tastatur Text eingeben kann. Der Benutzer wählt das Steuerelement und gibt ihm den Fokus, indem er es anklickt oder sich durch Benutzung der Tab -Taste zu ihm hinbewegt. Der Benutzer kann Text eingeben, wenn das Steuerelement ein blinkendes Caret anzeigt. Die Maus kann benutzt werden, um den Mauszeiger zu bewegen und Zeichen auszuwählen, die ersetzt werden sollen oder um den Mauszeiger zum Einfügen von Zeichen zu positionieren. Die Taste Backspace kann benutzt werden, um Zeichen zu löschen. Editierfelder verwenden die Systemschrift mit variablem Zeichenabstand und zeigen ANSI-Zeichen an. Anwendungen, die compiliert wurden, um unter früheren Versionen von Windows zu laufen, zeigen Text mit einer Systemschrift und festen Zeichenabständen an, wenn sie nicht durch das Windows 3.0 Dienstprogramm MARK mit der Option MEMORY FONT gekennzeichnet worden sind.

Klasse	Bedeutung
	Eine Anwendung kann auch die Botschaft *wm_SetFont* an das Editierfeld senden, um die voreingestellte Schrift zu wechseln. Editierfelder expandieren Tabulatorzeichen zu so vielen Leerzeichen, wie nötig sind, um den Mauszeiger zum nächsten Tabulatorstop zu bewegen. Es wird angenommen, daß Tabulatorstops in einem Abstand von acht Zeichen gesetzt sind.
ListBox	Bestimmt eine Liste von Strings. Dieses Steuerelement wird immer dann verwendet, wenn eine Anwendung eine Liste von Namen anzeigen soll, die der Benutzer anschauen und auswählen kann. Der Benutzer kann einen String auswählen, indem er auf ihn zeigt und klickt. Wenn ein String gewählt ist, wird er hervorgehoben dargestellt und eine Botschaft an das übergeordnete Fenster geschickt. Eine horizontale oder vertikale Bildlaufleiste kann verwendet werden, um Listen zu rollen, die zu lang für das Fenster sind. Die Liste macht die Bildlaufleisten automatisch sichtbar, wenn es notwendig ist.
MDIClient	Bestimmt ein MDI-Client-Fenster. Das MDI-Client-Fenster empfängt Botschaften, die die untergeordneten Fenster der MDI-Anwendung steuern. Die empfohlenen Stilbits sind *ws_ClipChildren* und *ws_Child*. Um ein rollbares MDI-Client-Fenster zu erzeugen, das dem Benutzer erlaubt, untergeordnete MDI-Fenster in den Sichtbereich zu rollen, kann eine Anwendung auch die Stile *ws_HScroll* und *ws_VScroll* benutzen.
ScrollBar	Bestimmt ein Rechteck, das ein Bildlaufsymbol enthält, und Richtungspfeile an beiden Enden hat. Die Bildlaufleiste schickt immer dann eine Benachrich tigungsbotschaft an ihr übergeordnetes Fenster, wenn der Benutzer das Steuerelement anklickt. Das über geordnete Fenster ist, wenn nötig, für das Aktualisieren der Position des Bildlaufsymbols verantwortlich. Bildlaufleistenelemente haben das gleiche Aussehen und die gleiche Funktion wie Bildlaufleisten, die in gewöhnlichen Fenstern verwendet werden. Im Gegensatz zu Bildlaufleisten können Bildlaufleistenelemente überall in einem Fenster positioniert werden, wann immer sie gebraucht werden, um rollbare Eingaben für ein Fenster bereitzustellen. Die Klasse *ScrollBar* schließt auch ein Größenveränderungssymbol ein. Ein Größenveränderungssymbol ist ein kleines Rechteck, das der Benutzer verwenden kann, um die Größe eines Fensters zu ändern.

Klasse	Bedeutung
Static	Bestimmt ein einfaches Textfeld, Kästchen oder Rechteck, das benutzt werden kann, um andere Steuerelemente zu beschriften, einzurahmen oder abzutrennen. Statische Steuerelemente nehmen keine Eingaben an und liefern keine Ausgaben.

Querverweis *RegisterClass, wm_Create, wm_GetMinMaxInfo, wm_NCCreate*

CreateWindowEx

Syntax
```
function CreateWindowEx(ExStyle: LongInt; ClassName,
WindowName: PChar; Style: LongInt; X, Y, Width, Height:
Integer; WndParent: Hwnd; Menu: HMenu; Instance:
THandle; Param: Pointer): HWnd;
```

Beschreibung Diese Funktion erzeugt ein überlapptes, ein Pop-up- oder ein untergeordnetes Fenster mit einem im Parameter *ExStyle* angegebenen erweiterten Stil.

Diese Funktion ist ansonsten identisch mit der Funktion *CreateWindow*. In der Beschreibung dieser Funktion finden Sie ausführliche Informationen über die Erzeugung eines Fensters.

ExStyle: Bestimmt den erweiterten Stil des Fensters, das erzeugt wird.

ClassName: Zeigt auf einen mit 0 abgeschlossenen String, der den Namen der Fensterklasse enthält.

WindowName: Zeigt auf einen mit 0 abgeschlossenen String, der den Fensternamen darstellt.

Style: Bestimmt den Stil des Fensters, das erzeugt wird.

X: Bestimmt die anfängliche x-Position des Fensters.

Y: Bestimmt die anfängliche y-Position des Fensters.

Width: Bestimmt die Breite des Fensters in Geräteeinheiten.

Height: Bestimmt die Höhe des Fensters in Geräteeinheiten.

WndParent: Bezeichnet das übergeordnete oder das besitzende Fenster des Fensters, das erzeugt wird.

Menu: Bestimmt den Bezeichner eines Menüs oder eines untergeordneten Fensters. Die Bedeutung ist vom Fensterstil abhängig.

Instance: Bezeichnet die Instanz des Moduls, das dem Fenster zugeordnet werden soll.

Param: Zeigt auf den Wert, der dem Fenster durch die *TCreateStruct*-Datenstruktur übergeben wird, auf die der Parameter *lParam* der Botschaft *wm_Create* zeigt.

Rückgabewert Der Rückgabewert bezeichnet das neue Fenster. Er ist 0, wenn das Fenster nicht erzeugt wird.

Hinweis Eine Beschreibung der erweiterten Fensterstile finden Sie im Abschnitt »ws_ex_Xxxx« in Kapitel 1 »Windows-Stile und -Konstanten«.

Die normalen Fensterstile sind ebenfalls in Kapitel 1 in den Abschnitten »ds_Xxxx«, »ws_Xxxx«, »bs_Xxxx«, »cbs_Xxxx«, »es_Xxxx«, »lbs_Xxxx«, »sbs_Xxxx« und »ss_Xxxx« beschrieben.

Querverweis *CreateWindow, wm_ParentNotify*

DebugBreak

Syntax
```
procedure DebugBreak;
```

Beschreibung Diese Prozedur erzwingt eine Unterbrechung zum Debugger. Diese Prozedur hat keine Parameter.

DefDlgProc

Syntax
```
function DefDlgProc(Dlg: HWnd; Msg, wParam: Word;
lParam: LongInt): LongInt;
```

Beschreibung Diese Funktion ermöglicht die Bearbeitung aller Windows-Botschaften, die von Dialogfenstern individueller Klassen nicht bearbeitet werden.

Alle Fensterbotschaften, die nicht explizit von der Fensterfunktion bearbeitet werden, müssen an *DefDlgProc*, nicht an *DefWindowProc* weitergegeben werden. Dies stellt sicher, daß Botschaften, die vom zugehörigen Fenster nicht bearbeitet werden können, trotzdem ordnungsgemäß abgearbeitet werden.

Dlg: Bezeichnet das Dialogfenster.

Msg: Bestimmt die Nummer der Botschaft.

wParam: Bestimmt 16 Bits zusätzlicher, botschaftsabhängiger Informationen.

lParam: Bestimmt 32 Bits zusätzlicher, botschaftsabhängiger Informationen.

Rückgabewert Der Rückgabewert hängt von der übergebenen Botschaft ab. Er bestimmt das Ergebnis der bearbeiteten Botschaft.

Hinweis

Anwendungen können Dialogfenster durch den Aufruf einer der folgenden Funktionen erstellen:

Funktion	Beschreibung
CreateDialog	Erzeugt ein nichtmodales Dialogfenster.
CreateDialogIndirect	Erzeugt ein nichtmodales Dialogfenster.
CreateDialogIndirectParam	Erzeugt ein nichtmodales Dialogfenster. und übergibt Daten an diese Fenster.
CreateDialogParam	Erzeugt ein nichtmodales Dialogfenster und übergibt Daten an dieses Fenster.
DialogBox	Erzeugt ein modales Dialogfenster.
DialogBoxIndirect	Erzeugt ein modales Dialogfenster.
DialogBoxIndirectParam	Erzeugt ein modales Dialogfenster und übergibt Daten an dieses Fenster.
DialogBoxParam	Erzeugt ein modales Dialogfenster und übergibt Daten an dieses Fenster.

DeferWindowPos

Syntax

```
function DeferWindowPos(WinPosInfo: THandle; Wnd,
WndInsertAfter: HWnd; x, y, cx, cy,: Integer; Flags:
Word): THandle;
```

Beschreibung

Diese Funktion aktualisiert die durch den Parameter *WinPosInfo* angegebene Datenstruktur für mehrere Fensterpositionen hinsichtlich des in *Wnd* gegebenen Fensters. Sie gibt das Handle der aktualisierten Struktur zurück. Die Funktion *EndDeferWindowPos* verwendet die Informationen in dieser Struktur, um Position und Größe mehrerer Fenster gleichzeitig zu verändern.

Die Struktur wird von der Funktion *BeginDeferWindowPos* erzeugt.

Die Parameter *x* und *y* bestimmen die neue Position, *cx* und *cy* die neue Größe des Fensters.

WinPosInfo: Bezeichnet eine Datenstruktur für mehrere Fensterpositionen, die Informationen über Größe und Position eines oder mehrerer Fenster enthält.

Die Struktur wird von der Funktion *BeginDeferWindowPos* oder vom letzten Aufruf der Funktion *DeferWindowPos* zurückgegeben.

Wnd: Bezeichnet das Fenster, dessen aktualisierte Daten in der Struktur abgelegt werden.

WndInsertAfter: Bezeichnet das Fenster, das dem zu aktualisierenden vorhergeht.

x: Bestimmt die x-Koordinate der oberen linken Ecke des Fensters.

y: Bestimmt die y-Koordinate der oberen linken Ecke des Fensters.

cx: Bestimmt die neue Breite des Fensters.

cy: Bestimmt die neue Höhe des Fensters.

Flags: Bestimmt einen von acht möglichen 16-Bit-Werten, die Größe und Position des Fensters beeinflussen. Möglich sind folgende Werte:

Wert	Bedeutung
swp_DrawFrame	Zeichnet den in der Fensterklassendefinition definierten Rahmen um das Fenster.
swp_HideWindow	Versteckt das Fenster.
swp_NoActivate	Fenster wird nicht aktiviert.
swp_NoMove	Erhält die aktuelle Position, ignoriert die Parameter *X und Y*.
swp_NoRedraw	Änderungen werden nicht dargestellt.
swp_NoSize	Erhält die aktuelle Größe, ignoriert die Parameter *cX* und *c*Y.
swp_NoZOrder	Erhält die aktuelle Ordnung, ignoriert den Parameter *WndInsertAfter.*
swp_ShowWindow	Stellt das Fenster dar.

Rückgabewert Der Rückgabewert bezeichnet die aktualisierte Datenstruktur für mehrere Fensterpositionen. Das zurückgegebene Handle kann von dem im Parameter *WinPosInfo* übergebenen abweichen. Das neue Handle sollte beim nächsten Aufruf an die Funktionen *DeferWindowPos* oder *EndDeferWindowPos* übergeben werden.

Der Rückgabewert ist dann 0, wenn die zur Verfügung stehenden Systemressourcen für die vollständige Bearbeitung dieser Funktion nicht ausreichen.

Hinweis Ist das Flag *swp_NoZOrder* nicht gesetzt, positioniert Windows das durch *Wnd* bezeichnete Fenster nach dem durch den Parameter *WndInsertAfter* festgelegte Fenster. Ist *WndInsertAfter* 0, wird das durch *Wnd* bezeichnete Fenster an den Anfang der Liste, ist der Parameter *1*, wird es an das Ende der Liste gesetzt.

Ist eines der Flags *swp_ShowWindow* oder *swp_HideWindow* gesetzt, können Bildlauf und Verschiebungen nicht gleichzeitig durchgeführt werden.

Die Koordinaten untergeordneter Fenster beziehen sich auf die linke obere Ecke des Client-Bereichs des übergeordneten Fensters.

Querverweis *BeginDeferWindowsPos, EndDeferWindowPos*

DefFrameProc

Syntax

```
function DefFrameProc(Wnd, WndMDIClient: HWnd; Msg,
wParam: Word; lParam: LongInt): LongInt;
```

Beschreibung

Diese Funktion unterstützt die normale Bearbeitung von Windows-Botschaften, die von der Fensterfunktion eines MDI-Rahmenfensters nicht bearbeitet werden. Alle Fensterbotschaften, die nicht explizit von der Fensterfunktion abgearbeitet werden, müssen an *DefFrameProc*, nicht an *DefWindowProc* weitergegeben werden.

Wnd: Bezeichnet das MDI-Rahmenfenster.

WndMDIClient: Bezeichnet das MDI-Client-Fenster.

Msg: Bestimmt die Nummer der Botschaft.

wParam: Bestimmt 16 Bits zusätzlicher, botschaftsabhängiger Informationen.

lParam: Bestimmt 32 Bits zusätzlicher, botschaftsabhängiger Informationen.

Rückgabewert

Der Rückgabewert hängt von der übergebenen Botschaft ab. Er bestimmt das Ergebnis der bearbeiteten Botschaft. Ist der Parameter *WndMDICClient* 0, entspricht der Rückgabewert dem der Funktion *DefWindowProc*.

Hinweis

Bearbeitet die Fensterprozedur einer Anwendung eine Botschaft nicht, so wird sie normalerweise an die Funktion *DefWindowProc* übergeben. MDI-Anwendungen verwenden zur voreingestellten Bearbeitung von Botschaften die Funktionen *DefFrameProc* und *DefMDIChildProc*.

Botschaften, die normalerweise an *DefWindowProc* weitergegeben würden, sollten stattdessen an *DefFrameProc* weitergegeben werden. Dies gilt für allgemeine Botschaften und für *wm_SetText*. Zudem können folgende Botschaften von *DefFrameProc* bearbeitet werden:

Botschaft	Standardbearbeitung von *DefFrameProc*
wm_Command	Das Rahmenfenster einer MDI-Anwendung erhält die Botschaft *wm_Command* zur Aktivierung eines bestimmten untergeordneten MDI-Fensters. Die gleichzeitig übergebene Fenster-ID ist der Bezeichner des durch Windows festgelegten untergeordneten MDI-Fensters. Begonnen wird mit dem ersten Bezeichner, der von einer Anwendung bei Erstellung des untergeordneten MDI-Fensters bestimmt wurde – der Wert darf sich nicht mit den Bezeichnern der Menüeinträge überschneiden.

Botschaft	Standardbearbeitung von *DefFrameProc*
wm_MenuChar	Wird die Tastenkombination Alt und Bindestrich - gedrückt, wird das Systemmenü des aktiven untergeordneten MDI-Fensters angewählt.
wm_NextMenu	Diese Botschaft veranlaßt die Auswahl des Systemmenüs des aktiven untergeordneten MDI-Fensters.
wm_SetFocus	*DefFrameProc* übergibt den Fokus an den MDI-Client, der ihn an das aktive untergeordnete MDI-Fenster weiterleitet.
wm_Size	Wenn die Prozedur des Rahmenfensters diese Botschaft an *DefFrameProc* weitergibt, so wird die Größe des untergeordneten MDI-Fensters an den neuen Client-Bereich angepaßt. Bringt die Prozedur des Rahmenfensters den MDI-Client auf eine abweichende Größe, sollte sie die Botschaft nicht an die Funktion *DefWindowProc* weitergeben.

DefHookProc

Syntax

```
function
lParam: LongInt; lpfnNextHook: TFarProc): LongInt;
```

Beschreibung

Diese Funktion ruft die folgende Funktion in einer Reihe von Filterverwaltungsfunktionen auf. Eine Filterverwaltungsfunktion bearbeitet Ereignisse, bevor sie an die botschaftsbearbeitende Schleife einer Anwendung in der Funktion *WinMain* übergeben wird.

Definiert eine Anwendung mehr als eine Filterverwaltungsfunktion mit Hilfe der Funktion *SetWindowsHook*, erstellt Windows eine verkettete Liste bzw. Filterverwaltungskette. Alle Funktionen desselben Typs werden jeweils in einer Kette positioniert.

Code: Bestimmt den Code der Windows-Filterverwaltungsfunktion (auch Botschaftsfilterfunktion). Diese bestimmt die Art der Bearbeitung der Botschaft.

wParam: Bestimmt den Word-Parameter der Botschaft, die von der Filterverwaltungsfunktion bearbeitet wird.

lParam: Bestimmt den Longint-Parameter der Botschaft, die von der Filterverwaltungsfunktion bearbeitet wird.

lpfnNextHook: Zeigt auf einen Speicherbereich, der die von der Funktion *SetWindowsHook* zurückgegebene Struktur *TFarProc* enthält. Windows ändert den Wert dieses Bereichs nach Aufruf der Funktion *UnhookWindowsHook*.

Rückgabewert Der Rückgabewert bestimmt einen Wert, der im direkten Verhältnis zum Parameter *Code* steht.

Querverweis *SetWindowsHook, UnhookWindowsHook*

DefineHandleTable

Syntax
```
function DefineHandleTable(Offset: Word): Bool;
```

Beschreibung Diese Funktion erzeugt eine individuelle Handle-Tabelle im Standarddatensegment einer Anwendung. In dieser Tabelle werden die von der Funktion *GlobalLock* zurückgegebenen Adressen globaler Speicherobjekte gespeichert. Im Real Mode aktualisiert Windows die zugehörige Adresse in der individuellen Handle-Tabelle, wenn das globale Speicherobjekt verschoben wird. Verwirft Windows ein Objekt mit dem zugehörigen Tabelleneintrag, ersetzt Windows die Adresse dieses Objekts in der Handle-Tabelle. Im Protected Mode (Standardmodus oder erweiterter 386-Modus) aktualisiert Windows die Adressen in der individuellen Handle-Tabelle nicht.

wOffset: Bestimmt den Offset vom Anfang des Datensegments zum Anfang der individuellen Handle-Tabelle. Hat es den Wert 0, bricht Windows die Aktualisierung der Handle-Tabelle ab.

Rückgabewert Nach erfolgreicher Ausführung der Funktion ist der Rückgabewert ungleich 0, sonst 0.

Hinweis Die individuelle Handle-Tabelle hat das folgende Format:

```
Count Clear_Number Entry[0] . . . Entry[Count-1]
```

Das erste Word (*Count*) in der Tabelle zeigt die Anzahl der Einträge an. Das zweite Word (*Clear_Number*) bestimmt die Anzahl der Einträge (gerechnet vom Anfang der Tabelle), die Windows auf 0 setzt, wenn die LRU-Speicherliste (least-recently-used) aktualisiert wird. Der Rest der Tabelle besteht aus einem Array, das die von *GlobalLock* zurückgegebenen Adressen enthält.

Anwendungen müssen das Feld *Count* in der Tabelle initialisieren, bevor sie *DefineHandleTable* aufrufen. Anwendungen können immer nur entweder das Feld *Count* oder das Feld *Clear_Number* verändern.

DefMDIChildProc

Syntax
```
function DefMDIChildProc(Wnd: HWnd; Msg, wParam: Word;
lParam: LongInt): LongInt;
```

Beschreibung Diese Funktion unterstützt die normale Bearbeitung von Windows-Botschaften, die von der Fensterfunktion eines unter-

geordneten MDI-Fensters nicht bearbeitet werden. Alle Fensterbotschaften, die nicht explizit von der Fensterfunktion abgearbeitet werden, müssen an die Funktion *DefMDIChildProc*, nicht an *DefWindowProc* weitergegeben werden.

Wnd: Bezeichnet das untergeordnete MDI-Fenster.

Msg: Bestimmt die Nummer der Botschaft.

wParam: Bestimmt 16 Bits zusätzlicher, botschaftsabhängiger Informationen.

lParam: Bestimmt 32 Bits zusätzlicher, botschaftsabhängiger Informationen.

Rückgabewert

Der Rückgabewert hängt von der übergebenen Botschaft ab. Er gibt das Ergebnis der bearbeiteten Botschaft an.

Hinweis

Die Funktion unterstellt, daß das übergeordnete Fenster des durch *Wnd* bezeichneten Fensters mit der Klasse *TMDIClient* erstellt wurde. Bearbeitet die Fensterprozedur einer Anwendung eine Botschaft nicht, so wird sie normalerweise an die Funktion *DefWindowProc* übergeben, die sie bearbeitet. MDI-Anwendungen verwenden zur Standardbearbeitung von Botschaften dagegen die Funktionen *DefFrameProc* und *DefMDIChildProc*. Botschaften, die normalerweise an die Funktion *DefWindowProc* weitergegeben würden, sollten stattdessen an *DefMDIChildProc* weitergegeben werden. Dies gilt für Botschaften aus dem Nicht-Client-Bereich und *wm_SetText*. Zudem werden folgende Botschaften von *DefMDIChildProc* bearbeitet:

Botschaft	Standardbearbeitung von *DefMDIChildProc*
wm_ChildActivate	Führt Aktivierungsprozeduren durch, wenn untergeordnete Fenster bewegt, dargestellt oder in der Größe verändert werden.
wm_GtMinMaxInfo	Berechnet die Größe eines vergrößerten untergeordneten MDI-Fensters auf Grundlage der aktuellen Größe des MDI-Client-Fensters.
wm_MenuChar	Übergibt die Taste an das Rahmenfenster.
wm_Move	Berechnet die Bildlaufleisten des MDI-Client-Bereichs neu, wenn diese existieren.
wm_NextMenu	Springt zur Menüleiste oder zum Systemmenü des Rahmens zurück.
wm_SetFocus	Aktiviert das untergeordnete Fenster, wenn es nicht das aktive MDI-Fenster ist.
wm_Size	Führt notwendige Operationen nach Veränderung der Größe (insbesondere Vergrößerung und Wiederherstellung) eines untergeordneten MDI-Fensters durch.

Botschaft	Standardbearbeitung von *DefMDIChildProc*
	Wird die Botschaft nicht an *DefMDIChildProc* übergeben, kann es zu unerwünschten Resultaten kommen.
wm_SysCommand	Bearbeitet auch den Befehl »Zum nächsten Fenster«.

DefWindowProc

Syntax
```
function DefWindowProc(Wnd: HWnd; Msg, wParam: Word;
lParam: LongInt): LongInt;
```

Beschreibung Diese Funktion unterstützt die Standardbearbeitung von Windows-Botschaften, die von der gegebenen Anwendung nicht bearbeitet werden. Alle Fensterbotschaften, die nicht explizit von der Klassenfensterfunktion abgearbeitet werden, müssen an *DefWindowProc* weitergegeben werden.

Wnd: Bezeichnet das Fenster, das die Botschaft weitergibt.

Msg: Bestimmt die Nummer der Botschaft.

wParam: Bestimmt 16 Bits zusätzlicher, botschaftsabhängiger Informationen.

lParam: Bestimmt 32 Bits zusätzlicher, botschaftsabhängiger Informationen.

Rückgabewert Der Rückgabewert hängt von der übergebenen Botschaft ab. Er zeigt das Ergebnis der bearbeiteten Botschaft an.

DeleteAtom

Syntax
```
function DeleteAtom(AnAtom: TAtom): TAtom;
```

Beschreibung Diese Funktion löscht ein Atom und – wenn der Referenzzähler des Atoms 0 ist – den zugehörigen String aus der Atom-Tabelle.

Der Referenzzähler eines Atoms bestimmt, wie oft ein Atom zur Atom-Tabelle hinzugefügt wurde. Die Funktion *AddAtom* erhöht den Zähler bei jedem Aufruf, die Funktion *DeleteAtom* vermindert den Zähler entsprechend. Der String wird von *DeleteAtom* nur dann gelöscht, wenn der Referenzzähler 0 ist.

AnAtom: Bezeichnet das zu löschende Atom und den zugehörigen nullterminierten String.

Rückgabewert Nach erfolgreicher Ausführung der Funktion wird 0 zurückgeliefert, ansonsten ist der Rückgabewert gleich dem Parameter *AnAtom*. In diesem Fall wird das Atom nicht gelöscht.

DeleteDC

Syntax
```
function DeleteDC(DC: HDC): Bool;
```

Beschreibung Diese Funktion löscht den angegebenen Gerätekontext. Entspricht der Parameter *DC* dem letzten Gerätekontext des gegebenen Geräts, wird dies dem Gerät mitgeteilt.

Alle Speicher- und Systemressourcen des Geräts werden freigegeben.

DC: Bezeichnet den Gerätekontext.

Rückgabewert Der Rückgabewert ist *True*, wenn der Gerätekontext erfolgreich gelöscht wurde, unabhängig davon, ob es sich um den letzten Kontext des Geräts handelt. Tritt ein Fehler auf, ist der Rückgabewert *False*.

Hinweis Um einen Gerätekontext zu löschen, dessen Handle die Anwendung durch einen Aufruf der Funktion *GetDC* erhalten hat, muß die Funktion *ReleaseDC* aufgerufen werden.

DeleteMenu

Syntax
```
function DeleteMenu(Menu: HMenu; Position, Flags:
Word): Bool;
```

Beschreibung Diese Funktion löscht einen Eintrag aus dem durch *Menu* bezeichneten Menü.

Gehört ein Pop-up-Menü zu diesem Eintrag, entfernt *DeleteMenu* auch das zugehörige Handle, der verwendete Speicher wird freigegeben.

Menu: Bezeichnet das zu ändernde Menü.

Position: Bestimmt den zu löschenden Menüeintrag. Ist *wFlags* auf *mf_ByPosition* gesetzt, bestimmt *Position* die Position des Menüeintrags.

Der erste Eintrag eines Menü hat die Position 0. Ist *Flags* auf *mf_ByCommand* gesetzt, bestimmt *Position* die Befehls-ID des Menüeintrags.

Flags: Legt fest, wie der Parameter *Position* interpretiert wird. Er kann auf *mf_ByCommand* (Vorgabe) oder auf *mf_ByPosition* gesetzt werden.

Rückgabewert Der Rückgabewert ist *True* nach erfolgreicher Ausführung, andernfalls *False*.

Hinweis Unabhängig davon, ob das Menü zu einem aktuell angezeigten Fenster gehört, sollte die Anwendung nach einer Veränderung des Menüs die Funktion *DrawMenuBar* aufrufen.

DeleteMetaFile

Syntax
```
function DeleteMetaFile(MF: THandle): Bool;
```

Beschreibung Diese Funktion verhindert den Zugriff auf ein Metafile durch Freigabe der zugehörigen Systemressourcen. Das Metafile selbst wird nicht entfernt, die Funktion macht das Handle des Metafiles (*MF*) ungültig. Der Zugriff auf das Metafile kann somit durch Erstellung eines neuen Handle mit Hilfe der Funktion *GetMetaFile* wiederhergestellt werden.

MF: Bezeichnet den zu löschenden Bezeichner.

Rückgabewert Der Rückgabewert bestimmt die Gültigkeit des Metafile-Handle. Er ist *True*, wenn die Systemressourcen freigegeben sind, *False*, wenn der Parameter *MF* kein gültiges Handle ist.

Querverweis *GetMetaFile*

DeleteObject

Syntax
```
function DeleteObject(Handle: THandle): Bool;
```

Beschreibung Diese Funktion löscht einen virtuellen Stift, Pinsel, Bitmap, eine Region, Schrift oder Farbpalette durch Freigabe des zugehörigen Systemspeichers. Nach Löschung des Objekts ist das Handle *Handle* ungültig.

Handle: Bezeichnet das Handle eines virtuellen Stifts, Pinsels, Bitmaps, einer Region, Schrift oder Farbpalette.

Rückgabewert Der Rückgabewert ist *True*, wenn das Objekt gelöscht wird. Er ist *False*, wenn der Parameter *Handle* kein gültiges Handle ist oder aktuell in einem Gerätekontext benutzt wird.

Hinweis Das zu löschende Objekt sollte aktuell nicht für einen Gerätekontext ausgewählt sein.

Wird ein Schraffurpinsel gelöscht, bleibt das zugehörige Bitmap unberührt, es muß selbständig gelöscht werden.

Anwendungen dürfen keinesfalls vordefinierte Objekte löschen.

DestroyCaret

Syntax
```
procedure DestroyCaret;
```

Beschreibung Diese Prozedur entfernt die Form des Carets, löst es vom aktuell verwaltenden Fenster und löscht es vom Bildschirm, wenn es sichtbar ist. Die Funktion *DestroyCaret* überprüft die Zugehörigkeit des Carets und löscht es nur, wenn es von einem Fenster der aktuellen Task verwaltet wird.

War die Form des Carets ein Bitmap, so wird dieses von *DestroyCaret* nicht freigegeben. Die Prozedur hat keine Parameter.

Hinweis Das Caret stellt eine gemeinsam genutzte Ressource dar. Hat ein Fenster eine bestimmte Form erzeugt, so wird diese gelöscht, bevor das Fenster den Fokus verliert oder deaktiviert wird.

DestroyCursor

Syntax
```
function DestroyCursor(Cursor: HCursor): Bool;
```

Beschreibung Diese Funktion entfernt den zuvor durch *CreateCursor* erzeugten Mauszeiger und gibt den belegten Speicherplatz frei. Sie sollte nicht zur Entfernung von Mauszeigern verwendet werden, die durch andere Funktionen als *CreateCursor* erzeugt wurden.

Cursor: Bezeichnet den zu entfernenden Mauszeiger. Der Mauszeiger darf aktuell nicht verwendet werden.

Rückgabewert Nach erfolgreicher Ausführung ist der Rückgabewert *True*, andernfalls ist er *False*.

Querverweis *CreateCursor*

DestroyIcon

Syntax
```
function DestroyIcon(Icon: HIcon): Bool;
```

Beschreibung Diese Funktion entfernt das zuvor durch *CreateIcon* erzeugte Symbol und gibt den belegten Speicherplatz frei. Sie sollte nicht zur Entfernung von Symbolen verwendet werden, die durch andere Funktionen als *CreateIcon* erzeugt wurden.

Icon: Bezeichnet das zu entfernende Symbol. Das Symbol darf aktuell nicht verwendet werden.

Rückgabewert Nach erfolgreicher Ausführung ist der Rückgabewert *True*, andernfalls ist er *False*.

Querverweis *CreateIcon*

DestroyMenu

Syntax
```
function DestroyMenu(Menu: HMenu): Bool;
```

Beschreibung Diese Funktion entfernt das durch *Menu* bezeichnete *Menü* und gibt den zugehörigen Speicher frei.

Menu: Bezeichnet das zu entfernende Menü.

Rückgabewert Der Rückgabewert ist *True*, wenn das Menü entfernt wurde, andernfalls *False*.

DestroyWindow

Syntax
```
function DestroyWindow(Wnd: HWnd): Bool;
```

Beschreibung Diese Funktion entfernt das angegebene Fenster. Sie versteckt oder schließt das Fenster dauerhaft, indem sie die entsprechenden Botschaften an das Fenster übergibt und den Fokus entfernt. Außerdem wird das Fenstermenü entfernt und freigegeben, die Anwendungs-Warteschlange geleert, ausstehende Timer gelöscht, der Besitz der Zwischenablage abgegeben und die Kette der Clipboard-Zwischenablage-Viewer unterbrochen, wenn das Fenster an deren Spitze steht. Die Funktion übergibt die Botschaften *wm_Destroy* und *wm_NCDestroy* an das Fenster.

Ist das gegebene Fenster anderen Fenstern übergeordnet, so werden seine untergeordneten Fenster ebenfalls entfernt. *DestroyWindow* entfernt zuerst die untergeordneten, dann das gegebene Fenster.

DestroyWindow entfernt auch nichtmodale Dialogfenster, die mit der Funktion *CreateDialog* erzeugt wurden.

Wnd: Bezeichnet das zu entfernende Fenster.

Rückgabewert Der Rückgabewert ist *True*, wenn das Fenster entfernt wurde, ansonsten ist er *False*.

DialogBox

Syntax
```
function DialogBox(Instance: THandle; TemplateName:
PChar; WndParent: HWnd; DialogFunc: TFarProc): Integer;
```

Beschreibung Diese Funktion erzeugt ein modales Dialogfenster mit dem Typ, der Größe und den Steuerelementen, die in der durch den Parameter *TemplateName* bezeichneten Dialogfenstervorlage beschrieben sind. Der Parameter *WndParent* bezeichnet das Anwendungsfenster, zu dem das Dialogfenster gehört. Botschaften, die das Dialogfenster empfängt, werden von der Callback-Funktion bearbeitet, auf die der Parameter *DialogFunc* zeigt.

Die Funktion *DialogBox* gibt die Kontrolle erst dann zurück, wenn die Callback-Funktion das modale Dialogfenster durch den Aufruf der Funktion *EndDialog* schließt.

Instance: Bezeichnet die Instanz des Moduls, dessen ausführbare Datei die Dialogfenstervorlage enthält.

TemplateName: Zeigt auf einen nullterminierten String, der die Dialogfenstervorlage benennt.

WndParent: Bezeichnet das Fenster, von dem das Dialogfenster verwaltet wird.

DialogFunc: Die Adresse der Prozedurinstanz der Dialogfunktion. Weitere Informationen im folgenden Hinweis.

Rückgabewert

Der Rückgabewert bestimmt den Wert des Parameters *Result* der Funktion *EndDialog*. Diese Funktion wird zum Schließen des Dialogfensters verwendet. Die Werte, die vom Dialogfenster zurückgegeben werden, werden von Windows bearbeitet und nicht an die Anwendung übergeben. Der Rückgabewert ist -1, wenn die Funktion das Dialogfenster nicht erstellen konnte.

Hinweis

Die Funktion *DialogBox* ruft die Funktion *GetDC* auf, um den Bildschirmkontext zu ermitteln. Es kann zu Problemen kommen, wenn alle Bildschirmkontexte im Bildschirmkontext-Cache von Windows durch *GetDC* ermittelt wurden und *DialogBox* auf einen weiteren Bildschirmkontext zuzugreifen versucht.

Ein Dialogfenster kann bis zu 255 Steuerelemente enthalten.

Callback-Funktion

```
function DialogFunc (hdlg: HWnd; wMsg, wParam: Word;
lParam: LongInt): Integer; export;
```

DialogFunc: Der Platzhalter für den anwendungsunterstützten Funktionsnamen.

hDlg: Bezeichnet das Dialogfenster, das die Botschaft empfängt.

wMsg: Bestimmt die Nummer der Botschaft.

wParam: Bestimmt 16 Bits zusätzlicher, botschaftsabhängiger Informationen.

lParam: Bestimmt 32 Bits zusätzlicher, botschaftsabhängiger Informationen.

Rückgabewert

Die Callback-Funktion sollte 0 zurückgeben, wenn die Funktion die Botschaft nicht bearbeitet, einen Wert ungleich 0, wenn sie bearbeitet wird.

Hinweis

Auch wenn die Callback-Funktion einer Fensterfunktion vergleichbar ist, darf sie nicht die Funktion *DefWindowProc* zur Bearbeitung unerwünschter Botschaften aufrufen. Diese Botschaften werden intern durch die Fensterfunktion der Dialogklasse bearbeitet.

Die Adresse der Callback-Funktion, die in dem Parameter *DialogFunc* übergeben wird, muß durch die Funktion *MakeProcInstance* erzeugt werden.

DialogBoxIndirect

Syntax

```
function DialogBoxIndirect(Instance, DialogTemplate:
THandle; WndParent: HWnd; DialogFunc: TFarProc):
Integer;
```

Beschreibung

Diese Funktion erzeugt ein modales Dialogfenster mit dem Typ, der Größe und den Steuerelementen, die in der durch den Parameter *TemplateName* bezeichneten Dialogfenstervorlage beschrieben sind. Der Parameter *WndParent* bezeichnet das Anwendungsfenster, welches das Dialogfenster besitzt. Botschaften, die das Dialogfenster empfängt, werden von der Callback-Funktion bearbeitet, auf die der Parameter *DialogFunc* zeigt.

DialogBoxIndirect gibt die Kontrolle erst dann zurück, wenn die Callback-Funktion das modale Dialogfenster durch Aufruf der Funktion *EndDialog* schließt.

Instance: Bezeichnet die Instanz des Moduls, dessen ausführbare Datei die Dialogfenstervorlage enthält.

DialogTemplate: Bezeichnet einen Block im globalen Speicher, der eine *TDlgTemplate*-Datenstruktur enthält.

WndParent: Bezeichnet das Fenster, zu dem das Dialogfenster gehört.

DialogFunc: Die Adresse der Prozedurinstanz der Dialogfunktion. Weitere Informationen im folgenden Hinweis .

Rückgabewert

Der Rückgabewert bestimmt den Wert des Parameters *Result* der Funktion *EndDialog*. Diese Funktion wird zum Schließen des Dialogfensters verwendet. Die Werte, die vom Dialogfenster zurückgegeben werden, werden von Windows bearbeitet und nicht an die Anwendung übergeben. Der Rückgabewert ist -1, wenn die Funktion das Dialogfenster nicht erstellen konnte.

Hinweis

Ein Dialogfenster kann bis zu 255 Steuerelemente enthalten.

Callback-Funktion

```
function DialogFunc (hdlg: HWnd; wMsg, wParam: Word;
lParam: LongInt): Integer; export;
```

DialogFunc: Der Platzhalter für den anwendungsunterstützten Funktionsnamen.

hDlg: Bezeichnet das Dialogfenster, das die Botschaft empfängt.

wMsg: Bestimmt die Nummer der Botschaft.

wParam: Bestimmt 16 Bits zusätzlicher, botschaftsabhängiger Informationen.

lParam: Bestimmt 32 Bits zusätzlicher, botschaftsabhängiger Informationen.

Rückgabewert Die Callback-Funktion sollte 0 zurückgeben, wenn die Funktion die Botschaft nicht bearbeitet, einen Wert ungleich 0, wenn sie bearbeitet wird.

Hinweis Auch wenn die Callback-Funktion einer Fensterfunktion vergleichbar ist, darf sie nicht die Funktion *DefWindowProc* zur Bearbeitung unerwünschter Botschaften aufrufen. Diese Botschaften werden intern durch die Fensterfunktion der Dialogklasse bearbeitet.

Die Adresse der Callback-Funktion, die in dem Parameter *DialogFunc* übergeben wird, muß durch die Funktion *MakeProcInstance* erzeugt werden.

DialogBoxIndirectParam

Syntax
```
function DialogBoxIndirectParam(Instance,
DialogTemplate: THandle; WndParent: HWnd; DialogFunc:
TFarProc; InitParam: LongInt): Integer;
```

Beschreibung Diese Funktion erzeugt ein modales Dialogfenster, übergibt vor dessen Darstellung die Botschaft *wm_InitDialog* an die Dialogfunktion und übergibt *InitParam* als Parameter *lParam* dieser Botschaft. Diese Botschaft erlaubt es der Dialogfunktion, die Steuerelemente des Dialogfensters zu initialisieren.

Weitere Informationen über die Erzeugung eines modalen Dialogfensters finden sich in der Beschreibung der Funktion *DialogBoxIndirect*.

Instance: Bezeichnet die Instanz des Moduls, dessen ausführbare Datei die Dialogfenstervorlage enthält.

DialogTemplate: Bezeichnet einen Block im globalen Speicher, der eine *TDlgTemplate*-Datenstruktur enthält.

WndParent: Bezeichnet das Fenster, zu dem das Dialogfenster gehört.

DialogFunc: Ist die Adresse der Prozedurinstanz der Dialogfunktion. Weitere Informationen finden sich unter der Beschreibung der Funktion *DialogBoxIndirect*.

InitParam: Ein 32-Bit-Wert, der von *DialogBoxIndirectParam* an die Dialogfunktion übergeben wird, wenn diese das Dialogfenster erstellt.

Rückgabewert Der Rückgabewert ist der Wert des Parameters *Result* der Funktion *EndDialog*. Diese Funktion wird zum Schließen des Dialogfensters verwendet. Die Werte, die vom Dialogfenster zurückgegeben werden, werden von Windows bearbeitet und

nicht an die Anwendung übergeben. Der Rückgabewert ist -1, wenn die Funktion das Dialogfenster nicht erstellen konnte.

DialogBoxParam

Syntax
```
function DialogBoxParam(Instance: THandle;
TemplateName: PChar; WndParent: HWnd; DialogFunc:
TFarProc; InitParam: LongInt): LongInt;
```

Beschreibung Diese Funktion erzeugt ein modales Dialogfenster, übergibt vor dessen Darstellung die Botschaft *wm_InitDialog* an die Dialogfunktion und *InitParam* als Parameter *lParam* dieser Botschaft. Die Botschaft erlaubt es der Dialogfunktion, die Steuerelemente des Dialogfensters zu initialisieren. Weitere Informationen über die Erzeugung eines modalen Dialogfensters finden sich in der Beschreibung der Funktion *DialogBox*.

Instance: Bezeichnet die Instanz des Moduls, dessen ausführbare Datei die Dialogfenstervorlage enthält.

TemplateName: Zeigt auf einen nullterminierten String, der die Dialogfenstervorlage benennt.

WndParent: Bezeichnet das Fenster, zu dem das Dialogfenster gehört.

DialogFunc: Ist die Adresse der Prozedurinstanz der Dialogfunktion. Weitere Informationen finden sich im Hinweis der Beschreibung der Funktion *DialogBox*.

InitParam: Ein 32-Bit-Wert, der von *DialogBoxParam* an die Dialogfunktion übergeben wird, wenn diese das Dialogfenster erstellt.

Rückgabewert Der Rückgabewert ist der Wert des Parameters *Result* der Funktion *EndDialog*. Diese Funktion wird zum Schließen des Dialogfensters verwendet. Die Werte, die vom Dialogfenster zurückgegeben werden, werden von Windows bearbeitet und nicht an die Anwendung übergeben. Der Rückgabewert ist -1, wenn die Funktion das Dialogfenster nicht erstellen konnte.

DispatchMessage

Syntax
```
function DispatchMessage(var Msg: TMsg): LongInt;
```

Beschreibung Diese Funktion übergibt die Botschaft in der durch *Msg* bezeichneten *TMsg*-Struktur an die Fensterfunktion des vorgegebenen Fensters.

Msg: Eine *TMsg*-Datenstruktur, die Botschaftsinformationen der Windows-Anwendungs-Warteschlange enthält. Die Struk-

tur muß gültige Botschaftswerte enthalten. Enthält *Msg* eine *wm_Timer*-Botschaft und ist der Parameter *lParam* dieser Botschaft nicht 0, ist dieser Parameter die Adresse einer Funktion, die anstelle der Fensterfunktion aufgerufen wird.

Rückgabewert Der Rückgabewert bestimmt den von der Fensterfunktion zurückgegebenen Wert. Er hängt von der übergebenen Botschaft ab, wird jedoch meist ignoriert.

DlgDirList

Syntax
```
function DlgDirList(Dlg: HWnd; PathSpec: PChar;
IDListBox, IDStaticPath: Integer; Filetype: Word):
Integer;
```

Beschreibung Diese Funktion füllt eine Liste mit Datei- oder Verzeichnisnamen. Die Liste wird durch den Parameter *IDListBox* bezeichnet und mit den Namen der Dateien gefüllt, die dem in *PathSpec* vorgegebenen Pfadnamen entsprechen.

DlgDirList zeigt Unterverzeichnisse in eckige Klammern eingeschlossen ([ ]) und Laufwerke in der Form [-x-], wobei x die Laufwerksbezeichnung darstellt.

Der Parameter *PathSpec* hat die folgende Form:

```
[[Laufwerk:]] [[ [[\u]]Verzeichnis
[[\Verzeichnis]]...\u]] [[Dateiname]]
```

In diesem Beispiel entspricht *Laufwerk:* der Laufwerksbezeichnung, *Verzeichnis* einem gültigen Verzeichnisnamen und *Dateiname* einem gültigen Dateinamen, der mindestens ein Platzhalterzeichen enthalten muß. Platzhalterzeichen sind das Fragezeichen (?, ersetzt ein Zeichen) und der Stern bzw. Asterisk (*, ersetzt beliebig viele Zeichen).

Schließt der Parameter *PathSpec* eine Laufwerksbezeichnung und/oder einen Verzeichnisnamen ein, werden das aktuelle Laufwerk und Verzeichnis entsprechend gewechselt, bevor die Liste gefüllt wird. Das durch *IDStaticPath* bezeichnete Textfeld wird ebenfalls mit der neuen Laufwerksbezeichnung und/oder dem neuen Verzeichnisnamen aktualisiert. Nach dem Füllen einer Liste wird *PathSpec* durch Entfernung der Laufwerksbezeichnung und/oder des Verzeichnisses aktualisiert.

DlgDirList sendet die Botschaften *lb_ResetContent* und *lb_dir* an die Liste.

Dlg: Bezeichnet das Dialogfenster, welches die Liste enthält.

PathSpec: Zeigt auf einen nullterminierten String mit dem Pfadnamen.

IDListBox: Bestimmt den Bezeichner einer Liste. Ist *IDListBox*

gleich 0, geht *DlgDirList* davon aus, daß die Liste nicht existiert und versucht nicht, sie zu füllen.

IDStaticPath: Bestimmt den Bezeichner des statischen Textfeldes, der für die Darstellung des aktuellen Laufwerks und Verzeichnisses verwendet wird. Ist *IDStaticPath* 0, geht *DlgDirList* davon aus, daß ein solches Texteld nicht existiert.

Filetype: Bestimmt die darzustellenden DOS-Dateiattribute. Möglich ist jede Kombination der im Hinweis beschriebenen Werte. Die Werte können mit Hilfe des bitweisen **or**-Operators kombiniert werden.

Rückgabewert Der Rückgabewert ist ungleich 0, wenn eine Liste dargestellt wurde, auch wenn es sich dabei um eine leere Liste handelt. Der Rückgabewert 0 zeigt an, daß der Eingabestring keinen gültigen Suchpfad enthielt. Der Parameter *Filetype* bestimmt die DOS-Dateiattribute der darzustellenden Dateien. Diese Attribute werden in der folgenden Tabelle beschrieben.

Attribut	Bedeutung
0x0000	Datendateien für Lese/Schreibzugriff ohne zusätzliche Attribute
0x0001	Schreibgeschützte Dateien
0x0002	Verborgene Dateien
0x0004	Systemdateien
0x0010	Unterverzeichnisse
0x0020	Archiv
0x2000	*lb_dir*-Flag (vgl. Anmerkung 1)
0x4000	Laufwerke
0x8000	Exklusiv-Bit (vgl. Anmerkung 2)

Anmerkungen: Ist das Flag *lb_dir* gesetzt, plaziert Windows die von *DlgDirList* generierten Botschaften in die Anwendungs-Warteschlange. Andernfalls werden sie unmittelbar an die Dialogfunktion übergeben.

Ist das Exklusiv-Bit gesetzt, werden nur Dateien des festgelegten Typs gelistet. Andernfalls werden die festgelegten Dateien zusätzlich zu den anderen Dateien angezeigt.

Querverweis *lb_ResetContent, lb_Dir*

DlgDirListComboBox

Syntax
```
function DlgDirListComboBox(Dlg: HWnd; PathSpec: PChar;
IDComboBox, IDStaticText: Integer; Filetype: Word):
Integer;
```

Beschreibung Diese Funktion füllt die Liste eines Kombinationsfensters mit Datei- oder Verzeichnisnamen.

Es wird die Liste des durch *IDComboBox* bezeichneten Kombinationsfensters mit den Namen der Dateien gefüllt, die der Dateinamensvorgabe *PathSpec* entsprechen.

Die Funktion *DlgDirListComboBox* zeigt Unterverzeichnisse in eckige Klammern eingeschlossen ([ ]) und Laufwerke in der Form [-x-], wobei x die Laufwerksbezeichnung darstellt.

Der Parameter *PathSpec* hat die folgende Form:

```
[[Laufwerk:]] [[ [[\u]]Verzeichnis[[\Verzeichnis]]
...\u]] [[Dateiname]]
```

Laufwerk: entspricht der Laufwerksbezeichnung, *Verzeichnis* einem gültigen Verzeichnisnamen und *Dateiname* einem gültigen Dateinamen, der mindestens ein Jokerzeichen enthalten muß. Jokerzeichen sind das Fragezeichen (?, ersetzt ein Zeichen) und der Stern (*, ersetzt beliebig viele Zeichen).

Schließt der Parameter *PathSpec* eine Laufwerksbezeichnung und/oder einen Verzeichnisnamen ein, werden das aktuelle Laufwerk und Verzeichnis entsprechend gewechselt, bevor die Liste gefüllt wird. Das durch *IDStaticText* bezeichnete Textelement wird ebenfalls mit der neuen Laufwerksbezeichnung und/oder dem neuen Verzeichnisnamen aktualisiert.

Nach dem Füllen der Liste eines Kombinationsfensters wird *PathSpec* durch Entfernung der Laufwerksbezeichnung und/oder des Verzeichnisses aktualisiert.

DlgDirListComboBox sendet die Botschaften *cb_ResetContent* und *cb_dir* an das Kombinationsfenster.

Dlg: Bezeichnet das Dialogfenster, welches das Kombinationsfenster enthält.

PathSpec: Zeigt auf einen nullterminierten String mit dem Verzeichnisnamen.

IDComboBox: Bestimmt den Bezeichner eines Kombinationsfensters. Ist *IDComboBox* gleich 0, geht *DlgDirListComboBox* davon aus, daß das Kombinationsfenster nicht existiert und versucht nicht, die Liste zu füllen.

IDStaticText: Bestimmt den Bezeichner des statischen Textfeldes, das für die Darstellung des aktuellen Laufwerks und Verzeichnisses verwendet wird. Ist er 0, geht *DlgDirListComboBox* davon aus, daß ein solches Textfeld nicht existiert.

Filetype: Bestimmt die darzustellenden DOS-Dateiattribute. Möglich ist jede Kombination der im Hinweis beschriebenen Werte.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn eine Liste dargestellt wurde, auch wenn es sich dabei um eine leere Liste handelt. Der Rückgabewert 0 zeigt an, daß der Eingabestring keinen gültigen Suchpfad enthielt.

Hinweis

Der Parameter *Filetype* bestimmt die DOS-Dateiattribute, diese Attribute werden in der folgenden Tabelle beschrieben.

Attribut	Bedeutung
0x0000	Datendateien für Lese/Schreibzugriff ohne zusätzliche Attribute
0x0001	Schreibgeschützte Dateien
0x0002	Verborgene Dateien
0x0004	Systemdateien
0x0010	Unterverzeichnisse
0x0020	Archiv
0x2000	*lb_dir*-Flag (vgl. Anmerkung 1)
0x4000	Laufwerke
0x8000	Exklusiv-Bit (vgl. Anmerkung 2)

Anmerkung 1: Ist das Flag *lb_dir* gesetzt, plaziert Windows die von *DlgDirList* generierten Botschaften in die Anwendungs-Warteschlange. Andernfalls werden sie unmittelbar an die Dialogfunktion übergeben.

Anmerkung 2: Ist das Exklusiv-Bit gesetzt, werden nur Dateien des festgelegten Typs gelistet. Andernfalls werden die festgelegten Dateien zusätzlich zu den anderen Dateien angezeigt.

Querverweis *cb_ResetContent, cb_Dir*

DlgDirSelect

Syntax

```
function DlgDirSelect(Dlg: HWnd; Str: PChar; IDListBox:
Integer): Bool;
```

Beschreibung

Diese Funktion ermittelt die aktuelle Auswahl in einer Liste. Es wird unterstellt, daß die Liste durch die Funktion *DlgDirList* gefüllt wurde und die Auswahl sich auf eine Laufwerksbezeichnung oder einen Datei- oder Verzeichnisnamen bezieht.

DlgDirSelect kopiert die Auswahl in den durch *Str* bezeichneten Puffer. Wenn sich die aktuelle Auswahl auf eine Laufwerksbezeichnung oder einen Verzeichnisnamen bezieht, entfernt *DlgDirSelect* die einschließenden eckigen Klammern (bei Laufwerksbezeichnungen zusätzlich die Bindestriche), so daß der Name in den neuen Pfadnamen eingefügt werden kann. Liegt keine Auswahl vor, wird *Str* nicht verändert.

DlgDirSelect übergibt die Botschaften *lb_GetCurSel* und *lb_Getext* an die Liste.

Dlg: Bezeichnet das Dialogfenster, das die Liste enthält.

Str: Zeigt auf einen Puffer, der den ausgewählten Pfadnamen aufnehmen soll.

IDListBox: Bestimmt die Integer-ID der Liste in einem Dialogfenster.

Rückgabewert Der Rückgabewert bestimmt den Status der aktuellen Auswahl in der Liste. Handelt es sich dabei um einen Verzeichnisnamen, ist der Rückgabewert *True,* andernfalls *False.*

Hinweis Die Funktion *DlgDirSelect* läßt nur die Rückgabe eines einzelnen Dateinamens aus der Liste zu.

Die Liste darf nicht für Mehrfachauswahl vorgesehen sein. Ist letzteres der Fall, wird nicht *True* zurückgegeben und außerdem bleibt der Parameter *Str* unverändert.

Querverweis *DlgDirList, lb_GetCurSel, lb_GetText*

DlgDirSelectComboBox

Syntax
```
function DlgDirSelectComboBox(Dlg: HWnd; Str: PChar;
IDComboBox: Integer): Bool;
```

Beschreibung Diese Funktion ermittelt die aktuelle Auswahl in der Liste eines Kombinationsfensters. Es wird unterstellt, daß die Liste durch *DlgDirListComboBox* gefüllt wurde und die Auswahl sich auf eine Laufwerksbezeichnung oder einen Datei- oder Verzeichnisnamen bezieht.

Die Funktion kopiert die Auswahl in den durch *Str* vorgegebenen Puffer. Bezieht sich die aktuelle Auswahl auf eine Laufwerksbezeichnung oder einen Verzeichnisnamen, so entfernt sie die einschließenden eckigen Klammern (bei Laufwerksbezeichnungen zusätzlich die Bindestriche), so daß der Name in den neuen Pfadnamen eingefügt werden kann. Liegt keine Auswahl vor, wird *Str* nicht verändert.

DlgDirSelectComboBox übergibt die Botschaften *cb_GetCurSel* und *cb_GetLBText* an das Kombinationsfenster.

Dlg: Bezeichnet das Dialogfenster, welches das Kombinationsfenster enthält.

Str: Zeigt auf einen Puffer, der den ausgewählten Pfadnamen entnehmen soll.

IDListBox: Bestimmt die Integer-ID des Kombinationsfensters in einem Dialogfenster.

Rückgabewert Der Rückgabewert bestimmt den Status der aktuellen Auswahl im Kombinationsfenster. Handelt es sich dabei um einen Verzeichnisnamen, ist der Rückgabewert *True*, andernfalls ist er *False.*

Hinweis Die Funktion *DlgDirSelectComboBox* läßt nur die Rückgabe eines einzelnen Dateinamens zu.

Querverweis *DlgDirListComboBox, cb_GetCurSel, db_GetText*

DOS3Call

Syntax
```
procedure DOS3Call;
```

Beschreibung Diese Prozedur ermöglicht Anwendungen den Aufruf des DOS-Interrupt 21H und kann anstelle eines direkt codierten DOS-Interrupt 21H verwendet werden. *DOS3Call* arbeitet unter Windows schneller als der direkte Zugriff auf den DOS-Interrupt. Diese Prozedur kann nur innerhalb eines Assemblerteils aufgerufen werden, da die Register vor dem Aufruf korrekt besetzt sein müssen. Man kann dazu eine als **external** deklarierte Routine verwenden oder den integrierten Assembler.

Vor Aufruf der Prozedur *DOS3Call* müssen alle Register wie für einen normalen INT 21H gesetzt werden. Am Ende der Funktion entsprechen alle Register der entsprechenden Funktion *INT 21H*.

Ein Aufruf könnte damit wie folgt aussehen:

```
var
  Version: Byte;
  .
  .
asm
   MOV   AH,$30
   Dos3Call
   MOV     Version,AL
end ;
WriteLn(Version);
  .
  .
```

DPtoLP

Syntax
```
function DPtoLP(DC: HDC; var Points; Count: Integer):
Bool;
```

Beschreibung Diese Funktion konvertiert die Punkte eines Geräts in virtuelle Punkte. Sie bildet die Koordinaten jedes durch den Parameter *Points* bezeichneten Punktes aus dem Koordinatensystem des Geräts im virtuellen Koordinatensystem der grafischen Geräteschnittstelle (GDI) ab. Die Konvertierung ist vom aktuellen Abbildungsmodus sowie dem Ursprung und der Ausdehnung von Gerätefenster und Grafikfenster abhängig.

DC: Bezeichnet den Gerätekontext.

Points: Zeigt auf ein Array von Punkten. Jeder Punkt muß eine *TPoint*-Datenstruktur sein.

Count: Bestimmt die Anzahl der Punkte im Array.

Rückgabewert Der Rückgabewert ist *True*, wenn alle Punkte konvertiert wurden, andernfalls *False*.

DrawFocusRect

Syntax
```
procedure DrawFocusRect(DC: HDC; var Rect: TRect);
```

Beschreibung Diese Prozedur zeichnet ein Rechteck im Fokusstil.

DC: Bezeichnet den Gerätekontext.

Rect: Eine *TRect*-Datenstruktur, die die Koordinaten des zu zeichnenden Rechtecks festlegt.

Hinweis Da es sich hier um eine **xor**-Prozedur handelt, löscht ein zweiter Aufruf dieser Prozedur das Rechteck vom Bildschirm.

In dem durch diese Prozedur gezeichneten Rechteck kann kein Bildlauf durchgeführt werden. Um einen Bildlauf in einem Bereich durchzuführen, der ein Rechteck enthält, das von dieser Prozedur gezeichnet wurde, muß das Rechteck mit Hilfe von *DrawFocusRect* vom Bildschirm gelöscht werden.

Nach erfolgtem Bildlauf muß *DrawFocusRect* erneut aufgerufen werden, um das Rechteck an der neuen Position zu zeichnen.

DrawIcon

Syntax
```
function DrawIcon(DC: HDC; X, Y: Integer; Icon: HIcon):
Bool;
```

Beschreibung Diese Funktion zeichnet ein Symbol im angegebenen Gerät. *DrawIcon* positioniert die obere linke Ecke des Symbols an der durch die Parameter *X* und *Y* bezeichneten Position.

Diese Position ist vom aktuellen Abbildungsmodus des Gerätekontextes abhängig.

DC: Bezeichnet den Gerätekontext eines Fensters.

X: Bestimmt die virtuelle x-Koordinate der oberen linken Ecke des Symbols.

Y: Bestimmt die virtuelle y-Koordinate der oberen linken Ecke des Symbols.

Icon: Bezeichnet das zu zeichnende Symbol.

Rückgabewert Nach erfolgreicher Ausführung der Funktion ist der Rückgabewert *True*, andernfalls *False*.

Hinweis Die Symbolressource muß vorab mit Hilfe der Funktion *LoadIcon* geladen werden. Vor dem Aufruf dieser Funktion muß der Abbildungsmodus *mw_Text* ausgewählt werden.

DrawMenuBar

Syntax
```
procedure DrawMenuBar(Wnd: HWnd);
```

Beschreibung Diese Prozedur stellt eine Menüleiste wieder her. Sie sollte insbesondere dann aufgerufen werden, wenn eine Menüleiste nach ihrer Erzeugung durch Windows verändert wurde.

Wnd: Bezeichnet das Fenster, dessen Menü wiederhergestellt werden muß.

DrawText

Syntax
```
function DrawText(DC: HDC; Str: PChar; Count: Integer;
var Rect: TRect; Format: Word): Integer;
```

Beschreibung Diese Funktion zeichnet formatierten Text in dem durch *Rect* bezeichneten Rechteck. Der Text wird durch Umwandlung von Tabulatoren in entsprechende Leerzeichen, linksbündige, rechtsbündige oder zentrierte Ausrichtung und einen für das Rechteck passenden Zeilenumbruch formatiert. Die Art der Formatierung wird durch den Parameter *Format* festgelegt.

DrawText verwendet zur Textdarstellung die für den Gerätekontext ausgewählte Schrift, Textfarbe und Hintergrundfarbe. Solange das *dt_NoClip*-Format nicht verwendet wird, kürzt *DrawText* den Text so, daß er nicht außerhalb des Rechtecks erscheint. Jede Formatierung geht von mehreren Textzeilen aus, außer wenn das Format *dt_SingleLine* vorgegeben ist.

DC: Bezeichnet den Gerätekontext.

Str: Zeigt auf den darzustellenden String. Hat der Parameter *Count* den Wert -1, muß der String nullterminiert sein.

Count: Bestimmt die Anzahl der Bytes im String. Ist *Count* -1, wird davon ausgegangen, daß *Str* ein Zeiger auf einen nullterminierten String ist. In diesem Fall berechnet *DrawText* die Anzahl der Zeichen automatisch.

Rect: Zeigt auf eine *TRect*-Datenstruktur, die das Rechteck, in dem der Text formatiert werden soll, in virtuellen Koordinaten enthält.

Format: Bestimmt die Art der Textformatierung. Möglich ist jede Kombination der in Kapitel 1 »Windows-Stile und -Konstanten« in Abschnitt »dt_Xxxx« beschriebenen Werte.

Rückgabewert Der Rückgabewert bestimmt die Höhe des Textes.

Hinweis Auch wenn die ausgewählte Schriftart zu groß für das vorgegebene Rechteck ist, wird sie von der Funktion *DrawText* nicht durch eine kleinere ersetzt.

Der Abschnitt »dt_Xxxx« in Kapitel 1 »Windows-Stile und -Konstanten« beschreibt die Werte des Parameters *Format*. Diese Werte können mit Hilfe des bitweisen **or**-Operators kombiniert werden. Zu beachten ist, daß die Werte *dt_CaleRect*, *dt_ExternalLeading*, *dt_Internal*, *dt_NoClip* und *dt_NoPrefix* nicht zusammen mit dem Wert *dt_TabStop* verwendet werden können.

Ellipse

Syntax
```
function Ellipse(DC: HDC; X1, Y1, X2, Y2: Integer): Bool;
```

Beschreibung Diese Funktion zeichnet eine Ellipse. Der Mittelpunkt der Ellipse wird durch den Mittelpunkt des eingrenzenden Rechtecks – bestimmt durch die Parameter *X1*, *Y1*, *X2* und *Y2* – festgelegt. Der Rahmen der Ellipse wird mit dem aktuellen Stift gezeichnet. Das Innere der Ellipse wird mit dem aktuellen Pinsel ausgefüllt. Ist das eingrenzende Rechteck leer, wird nichts gezeichnet.

DC: Bezeichnet den Gerätekontext.

X1: Bestimmt die virtuelle x-Koordinate der oberen linken Ecke.

Y1: Bestimmt die virtuelle y-Koordinate der oberen linken Ecke.

X2: Bestimmt die virtuelle x-Koordinate der unteren rechten Ecke.

Y2: Bestimmt die virtuelle y-Koordinate der unteren rechten Ecke.

Rückgabewert Der Rückgabewert ist *True*, wenn die Ellipse gezeichnet wurde, andernfalls *False*.

Hinweis Die Breite des Rechtecks, die durch die Differenz *X2 - X1* festgelegt ist, darf 32 767 Einheiten nicht übersteigen. Dieses Einschränkung gilt auch für die Höhe des Rechtecks (*Y2 - Y1*).

Die aktuelle Position wird durch diese Funktion weder benutzt noch verändert.

EmptyClipboard

Syntax
```
function EmptyClipboard: Bool;
```

Beschreibung Diese Funktion leert die Zwischenablage und gibt die Handles auf die entsprechenden Daten frei, außerdem wird der Besitz der Zwischenablage dem aufrufenden Fenster zugewiesen.

Diese Funktion hat keine Parameter.

Rückgabewert Der Rückgabewert ist *True*, wenn die Zwischenablage leer ist, ist er *False*, liegt ein Fehler vor.

Hinweis Die Zwischenablage muß beim Aufruf der Funktion *EmptyClipboard* geöffnet sein.

EnableHardwareInput

Syntax
```
function EnableHardwareInput(EnableInput: Bool): Bool;
```

Beschreibung Diese Funktion schaltet Maus- und Tastatureingaben aus. Die Eingaben werden gespeichert, wenn der Parameter *bEnableInput* den Wert *True* hat, ist er *False*, werden die Eingaben verworfen.

EnableInput: Veranlaßt die Speicherung der Eingaben, wenn der Parameter *EnableInput* den Wert *True* hat, ist er *False*, werden die Eingaben verworfen.

Rückgabewert Der Rückgabewert zeigt an, ob die Eingabe über Maus und Tastatur abgeschaltet ist. Er ist *True* (Standardrückgabewert), wenn die Eingabe vorher eingeschaltet war, andernfalls *False*.

EnableMenuItem

Syntax
```
function EnableMenuItem(Menu: HMenu; IDEnableItem:
Word; Enable: Word): Bool;
```

Beschreibung Diese Funktion aktiviert oder deaktiviert Menüoptionen oder versetzt sie in den nicht aktivierbaren Zustand.

Menu: Bestimmt das Menü.

IDEnableItem: Bestimmt die zu markierende Menüoption. Der Parameter *IDEnableItem* kann sich auf Menüpunkte aus Zeilen- und Pop-Up-Menüs beziehen.

Enable: Legt die zugehörige Aktion fest. Dies kann eine Kombination von *mf_Disabled*, *mf_Enabled* oder *mf_Grayed* mit *mf_ByCommand* oder *mf_ByPosition* sein. Diese Werte können mit Hilfe des bitweisen **or**-Operators kombiniert werden. Die Werte haben folgende Bedeutung:

Wert	Bedeutung
mf_ByCommand	Legt fest, daß der Parameter *IDEnableItem* die Menüpunkt-ID angibt (*mf_ByCommand* ist die Standard-ID).
mf_ByPosition	Legt fest, daß der Parameter *IDEnableItem* die Position des Menüpunkts angibt (der erste Menüpunkt liegt an der Position 0).

Wert	Bedeutung
mf_Disabled	Menüpunkt ist deaktiviert.
mf_Enabled	Menüpunkt ist aktiviert.
mf_Grayed	Menüpunkt ist in nicht aktivierbarem Zustand.

Rückgabewert Der Rückgabewert definiert den vorhergehenden Menüstatus, er ist -1, wenn der Menüpunkt nicht existiert.

Hinweis In der Beschreibung der Botschaft *wm_SysCommand* wird erläutert, wie Eingaben in eine Menüleiste verhindert oder ermöglicht werden.

EnableWindow

Syntax
```
function EnableWindow(Wnd: HWnd; Enable: Bool): Bool;
```

Beschreibung Diese Funktion aktiviert bzw. deaktiviert die Eingabe über Maus und Tastatur in das festgelegte Fenster oder Steuerelement. Ist die Eingabe deaktiviert, werden Mausklicks und gedrückte Tasten vom Fenster ignoriert, ist sie aktiviert, werden alle Eingaben bearbeitet. *EnableWindow* aktiviert Maus- und Tastatureingaben für das angegebene Fenster, wenn der Parameter *Enable True* ist, ist er *False*, werden die Eingaben deaktiviert.

Wnd: Bezeichnet das betroffene Fenster.

Enable: Bestimmt, ob das angegebene Fenster aktiviert oder deaktiviert werden soll.

Rückgabewert Der Rückgabewert ist *True*, wenn das Fenster korrekt aktiviert bzw. deaktiviert wurde, ergibt sich ein Fehler, ist er *False*.

Hinweis Ein Fenster muß aktiviert werden, bevor es angewählt werden kann. Zeigt z.B. eine Anwendung ein nichtmodales Dialogfenster an und das Hauptfenster ist deaktiviert, muß das Hauptfenster vor dem Löschen des Dialogfensters aktiviert werden, sonst erhält ein anderes Fenster den Fokus. Ist ein untergeordnetes Fenster deaktiviert, wird es von Windows ignoriert, wenn es festzustellen versucht, welches Fenster Mausbotschaften erhalten soll. Anfangs sind alle Fenster aktiviert. *EnableWindow* dient zur expliziten Deaktivierung eines Fensters.

EndDeferWindowPos

Syntax
```
procedure EndDeferWindowPos(WinPosInfo: THandle);
```

Beschreibung Diese Prozedur aktualisiert in einem einzigen Durchlauf gleichzeitig Position und Größe eines oder mehrerer Fenster

auf dem Bildschirm. Der Parameter *WinPosInfo* bezeichnet eine Datenstruktur mit mehreren Fensterpositionen, in der die Aktualisierungsdaten für die Fenster abgelegt sind.

Die Funktion *DeferWindowPos* legt die Aktualisierungsdaten in der Datenstruktur ab, die Funktion *BeginDeferWindowPos* erzeugt zu Beginn die von den anderen Funktionen verwendete Datenstrukur.

WinPosInfo: Bezeichnet eine Datenstruktur für Fensterpositionen mit Informationen über Größe und Position eines oder mehrerer Fenster.

Diese Struktur wird von der Funktion *BeginDeferWindowPos* oder vom letzten Aufruf der Funktion *DeferWindowPos* zurückgegeben.

EndDialog

Syntax
```
procedure EndDialog(Dlg: HWnd; Result: Integer);
```

Beschreibung Die Prozedur schließt ein modales Dialogfenster und übergibt die Eingabewerte an die Funktion *DialogBox*, die das Dialogfenster erzeugt hat. *EndDialog* muß die Bearbeitung immer abschließen, wenn ein modales Dialogfenster mit der Funktion *DialogBox* erzeugt wurde. Sie muß in der Dialogfunktion eines modalen Dialogfensters verwendet werden. Andere Aufrufe der Funktion sind nicht vorgesehen.

Die Dialogfunktion kann *EndDialog* jederzeit aufrufen, auch während der Abarbeitung der Botschaft *wm_InitDialog*. Wird sie während der Bearbeitung dieser Botschaft aufgerufen, so wird das Dialogfenster geschlossen, bevor es sichtbar ist bzw. bevor der Fokus gesetzt wird.

EndDialog schließt das Dialogfenster nicht sofort, sondern setzt ein Flag, das das Fenster anweist, unmittelbar nach Beendigung der Dialogfunktion zu schließen. *EndDialog* übergibt die Steuerung an die aufrufende Dialogfunktion, die ihrerseits an Windows übergeben muß.

Dlg: Bezeichnet das zu schließende Dialogfenster.

Result: Enthält den Wert, der vom Dialogfenster an die aufrufende Funktion *DialogBox* zurückgegeben wird.

EndPaint

Syntax
```
procedure EndPaint(Wnd: Hwnd; var Paint: TPaintStruct);
```

Beschreibung Die Prozedur legt das Ende des Zeichnens im aktuellen Fenster fest. *EndPaint* ist nach jedem Aufruf der Funktion *BeginPaint*, aber erst nach Vollendung der Zeichnung, erforderlich.

Wnd: Legt das neu zu zeichnende Fenster fest.

Paint: Eine *TPaintStruct*-Datenstruktur, in der die durch *BeginPaint* ermittelten Zeichnungsinformationen abgelegt sind.

Hinweis — Wurde das Caret von der Funktion *BeginPaint* verborgen, wird es von *EndPaint* wiederhergestellt.

EnumChildWindows

Syntax

```
function EnumChildWindows(WndParent: HWnd; EnumFunc:
TFarProc; lParam: LongInt): Bool;
```

Beschreibung — Diese Funktion listet die untergeordneten Fenster des festgelegten übergeordneten Fensters auf, indem das Handle jedes untergeordneten Fensters an die anwendungsunterstützte Callback-Funktion übergeben wird. Die Callback-Funktion wird durch den Zeiger *EnumFunc* bezeichnet.

Die Funktion *EnumChildWindows* zählt die untergeordneten Fenster auf, bis die aufgerufene Funktion den Wert 0 zurückgibt oder das letzte untergeordnete Fenster aufgelistet wurde.

WndParent: Bestimmt das übergeordnete Fenster, dessen untergeordnete Fenster aufgelistet werden sollen.

EnumFunc: Ist die Adresse der Prozedurinstanz der Callback-Funktion.

lParam: Legt den Wert fest, der an die Callback-Funktion zur Verwendung in der Anwendung übergeben wird.

Rückgabewert — Wurden alle untergeordneten Fenster aufgelistet, ist der Rückgabewert *True*, andernfalls ist er *False*.

Hinweis — Pop-Up-Fenster, die zum Parameter *WndParent* gehören, werden nicht aufgelistet.

Die als *EnumFunc* übergebene Adresse muß zuvor durch die Funktion *MakeProcInstance* erzeugt worden sein.

Callback-Funktion

```
function EnumFunc (Wnd: HWnd; lParam: LongInt): Bool;
export;
```

Beschreibung — *EnumFunc:* Der Platzhalter für den anwendungsunterstützten Funktionsnamen.

hWnd: Bezeichnet das Handle des Fensters.

lParam: Legt das LongInt-Argument der Funktion *EnumChildWindows* fest.

Rückgabewert — Soll die Auflistung fortgesetzt werden, sollte der Rückgabewert der Callback-Funktion *True* sein. Ein Rückgabewert von 0 stoppt die Auflistung.

EnumClipboardFormats

Syntax
```
function EnumClipboardFormats(Format: Word): Word;
```

Beschreibung Die Funktion listet die zur Zwischenablage gehörigen Formate aus einer Liste der verfügbaren Formate auf. In jedem Funktionsaufruf verweist der Parameter *Format* auf ein bekanntes, verfügbares Format. Zurückgegeben wird das nächste Format aus der Liste. Wird *Format* auf 0 gesetzt, wird das erste Format der Liste ausgegeben.

Format: Legt ein bekanntes Format fest.

Rückgabewert Der Rückgabewert kennzeichnet das nächste bekannte Datenformat der Zwischenablage. Er ist 0, wenn *Format* den letzten Eintrag in der Liste bezeichnet oder die Zwischenablage nicht geöffnet ist.

Hinweis Bevor die Formate durch *EnumClipboardFormats* aufgelistet werden können, muß die Zwischenablage mit der Funktion *OpenClipboard* geöffnet werden.

Die Reihenfolge, in der die sendende Anwendung alternative Formate für identische Daten in der Zwischenablage abgelegt hat, entspricht der Reihenfolge, mit der die Auflistung die Formate an die empfangende Anwendung übergibt. Diese Anwendung sollte das erste gelistete Format verwenden, mit dem sie arbeiten kann. Auf diese Weise kann der Sender die Formate empfehlen, die den geringsten Datenverlust zur Folge haben.

Querverweis *OpenClipboard*

EnumFonts

Syntax
```
function EnumFonts(DC: HDC; FaceName: PChar; FontFunc:
TFarProc; Data: Pointer): Integer;
```

Beschreibung Die Funktion listet die auf einem bestimmten Gerät zur Verfügung stehenden Schriften auf. Sie findet für jede Schrift, deren Schriftartname durch den Parameter *FaceName* festgelegt wurde, die zugehörigen Informationen und übergibt sie an die Funktion, auf die der Parameter *FontFunc* verweist.

Die anwendungsunterstützte Callback-Funktion kann nun die Schriftinformationen wie gewünscht bearbeiten. Die Auflistung wird fortgesetzt, bis keine Schriften mehr gefunden werden oder die Callback-Funktion 0 zurückgibt.

DC: Bezeichnet den Gerätekontext.

FaceName: Zeigt auf den nullterminierten String, der den Schriftartnamen der gewünschten Schriften festlegt. Ist *Face-*

Name 0, wählt *EnumFonts* zufällig aus und listet eine Schrift für jede verfügbare Schriftart auf.

FontFunc: Ist die Adresse der Prozedurinstanz der *Callback*-Funktion. Einzelheiten hierzu finden Sie im »Hinweis«.

Data: Zeigt auf die von der Anwendung zur Verfügung gestellten Daten. Die Daten werden zusammen mit den Schriftinformationen an die Callback-Funktion übergeben.

Rückgabewert

Der Rückgabewert ist der letzte Rückgabewert der Callback-Funktion. Seine Bedeutung ist anwenderdefiniert.

Hinweis

Die Adresse, die im Parameter *FontFunc* übergeben wird, muß durch die Funktion *MakeProcInstance* erzeugt werden.

Callback-Funktion

```
function FontFunc (LogFont: TLogFont; TextMetrics:
TTextMetric; nFontType: Integer; lpData: PChar):
Integer; export;
```

Beschreibung

FontFunc: Platzhalter für den anwendungsunterstützten Funktionsnamen.

lpLogFont: Zeigt auf eine *TLogFont*-Datenstruktur, die Informationen über die logischen Attribute der Schrift enthält.

lpTextMetrics: Zeigt auf eine *TTextMetric*-Datenstruktur, die Informationen über die physikalischen Attribute der Schrift enthält.

nFontType: Legt die Schriftart fest.

lpData: Zeigt auf die anwendungsunterstützen Daten, die von *EnumFonts* übergeben werden.

Rückgabewert

Rückgabewert kann jeder Integer-Wert sein.

Hinweis

Der Operator **and** kann zusammen mit den Konstanten *Raster_FontType* und *Device_FontType* zur Bestimmung der Schriftart verwendet werden. Das *Raster_FontType*-Bit des Parameters *nFontType* legt fest, ob es sich um eine Bitmap- oder eine Vektor-Schrift handelt. Hat das Bit den Wert 1, handelt es sich um eine Bitmap-Schrift, ist der Wert 0, liegt eine Vektor-Schrift vor.

Das *Device_FontType*-Bit unterscheidet Geräteschriften und Schriften der grafischen Geräteschnittstelle (GDI). Ist der Wert 1, liegt eine Geräteschrift, ist er 0 liegt eine GDI-Schrift vor.

Kann das Gerät Texttransformationen (skalieren, kursiv usw.) durchführen, wird nur die grundlegende Schrift gelistet. Der Anwender muß prüfen, welche Texttransformationen das jeweilige Gerät durchführen kann, um zu entscheiden, welche zusätzlichen Schriften direkt über das Gerät abzurufen sind. GDI kann die Attribute fett, kursiv, unterstrichen und durchgestrichen für jede GDI-Schrift emulieren.

EnumFonts listet nur Schriften aus der internen GDI-Tabelle auf. Schriften, die durch das Gerät selbst erzeugt werden, sind nicht eingeschlossen. Die Möglichkeiten eines Geräts zur Texttransformation können mit der Funktion *GetDeviceCaps* ermittelt werden. Die Information ist über den Index *TextCaps* verfügbar.

GDI kann GDI-Bitmap-Schriften horizontal im Verhältnis eins zu fünf und vertikal im Verhältnis eins zu acht skalieren, außer wenn *Proof-Quality* benutzt wird.

EnumMetaFile

Syntax

```
function EnumMetaFile(DC: HDC; MF: THandle;
CallbackFunc: TFarProc; ClientData: LPByte): Bool;
```

Beschreibung

Diese Funktion listet die GDI-Aufrufe innerhalb des durch den Parameter *MF* bezeichneten Metafiles auf. Sie übergibt jeden GDI-Aufruf innerhalb des Metafiles an die durch den Parameter *CallbackFunc* bezeichnete Funktion. Diese anwendungsunterstützte Callback-Funktion kann jeden GDI-Aufruf in gewünschter Weise abarbeiten. Die Auflistung wird fortgesetzt, bis keine GDI-Aufrufe mehr vorhanden sind oder die Callback-Funktion den Wert 0 zurückgibt.

DC: Bezeichnet das dem Metafile zugeordnete Gerät.

MF: Bezeichnet das Metafile.

CallbackFunc: Ist die Adresse der Prozedurinstanz der Callback-Funktion. Einzelheiten finden Sie im folgenden »Hinweis«.

ClientData: Zeigt auf die Daten der Callback-Funktion.

Rückgabewert

Der Rückgabewert ist *True*, wenn die Callback-Funktion alle GDI-Aufrufe in einem Metafile auflistet, andernfalls hat er den Wert *False*.

Callback-Funktion

```
function EnumFunc (DC: HDC; lpHTable: LPHandle; lpMFR:
PMetaRecord; nObj: Integer; lpClientDate: PByte):
Integer; export;
```

Beschreibung

EnumFunc: Platzhalter für den anwendungsunterstützten Funktionsnamen.

hDC: Der das Metafile beinhaltende Gerätekontext.

lpHTable: Zeigt auf eine Handle-Tabelle mit den zugehörigen Objekten (Stifte, Pinsel usw.) im Metafile.

lpMFR: Zeigt auf einen Metafile-Record im Metafile.

nObj: Bestimmt die Anzahl der Objekte und ihrer Handles in der Handle-Tabelle.

lpClientData: Zeigt auf die von der Anwendung bereitgestellten Daten.

Rückgabewert Die Funktion muß *True* zurückgeben, wenn die Auflistung fortgesetzt. Sie muß *False*zurückgeben, wenn sie beendet werden soll.

EnumObjects

Syntax
```
function EnumObjects(DC: HDC; ObjectType: Integer;
ObjectFunc: TFarProc; Data: Pointer): Integer;
```

Beschreibung Diese Funktion listet die verfügbaren Stifte oder Pinsel eines Geräts auf.

Die Callback-Funktion erhält die Informationen für jedes Objekt des vorgegebenen Typs. Die Callback-Funktion wird aufgerufen, bis keine weiteren Objekte mehr auffindbar sind oder der Rückgabewert 0 ist.

DC: Bezeichnet den Gerätekontext.

ObjectType: Bestimmt den Objekttyp. Möglich sind die Werte *obj_Brush* und *obj_Pen*.

ObjectFunc: Ist die Adresse der Prozedurinstanz der Callback-Funktion. Einzelheiten hierzu finden Sie im folgenden Hinweis.

Data: Zeigt auf die von der Anwendung zur Verfügung gestellten Daten. Die Daten werden zusammen mit den Objektinformationen an die Callback-Funktion übergeben.

Rückgabewert Der Rückgabewert ist der letzte Rückgabewert der Callback-Funktion. Die Bedeutung dieses Wertes wird vom Benutzer definiert.

Hinweis Die Adresse, die durch den Parameter *ObjectFunc* übergeben wird, muß durch die Funktion *MakeProcInstance* erzeugt werden.

Callback-Funktion
```
function ObjectFunc (LpLogObject, lpData: PChar):
Integer; export;
```

Beschreibung *ObjectFunc:* Der Platzhalter für den anwendungsunterstützten Funktionsnamen.

lpLogObject: Zeigt auf eine *TLogPen*- oder *TLogBrush*-Datenstruktur, die Informationen über die virtuellen Attribute des Objekts enthält.

lpData: Zeigt auf die Daten der Anwendung, die an die Funktion *EnumObjects* übergeben werden.

EnumProps

Syntax

```
function EnumProps(Wnd: HWnd; EnumFunc: TFarProc):
Integer;
```

Beschreibung

Diese Funktion listet alle Einträge der Eigenschaftsliste des betroffenen Fensters auf. Die Auflistung erfolgt durch Übergabe jedes einzelnen Eintrags an die durch *EnumFunc* bezeichnete Callback-Funktion. *EnumProps* läuft solange, bis alle Einträge aufgelistet wurden oder die Callback-Funktion 0 zurückgibt.

Wnd: Legt das Fenster fest, dessen Eigenschaften aufgelistet werden sollen.

EnumFunc: Ist die Adresse der Prozedurinstanz der *Callback-Funktion*. Einzelheiten hierzu finden Sie im folgenden Hinweis.

Rückgabewert

Der Rückgabewert ist der letzte Rückgabewert der Callback-Funktion. Er ist -1, wenn die Funktion keine Liste mit Eigenschaften findet.

Hinweis

Eine Anwendung sollte nur Eigenschaften löschen, die von ihr selbst hinzugefügt wurden. Eigenschaftseinträge, die durch andere Anwendungen oder Windows hinzugefügt wurden, sollten keinesfalls gelöscht werden.

Für die Callback-Funktion gelten folgende Einschränkungen:

1. Die Callback-Funktion darf die Steuerung weder abgeben noch Abläufe aufrufen, die die Steuerung an andere Tasks abgeben könnten.

2. Der Aufruf der Funktion *RemoveProp* in der Callback-Funktion ist möglich. Die Funktion *RemoveProp* kann jedoch nur Listeneinträge tilgen, die durch die Parameter der Callback-Funktion an diese übergeben wurden.

3. Eine Callback-Funktion sollte nicht versuchen, Listeneinträge hinzuzufügen.

Die Adresse, die durch den Parameter *EnumFunc* übergeben wird, muß durch die Funktion *MakeProcInstance* erzeugt werden.

Callback-Funktion

```
function EnumFunc(Wnd: HWnd; lpString: PChar; hData:
Thandle): Integer; export;
```

In Anwendungen und dynamischen Bibliotheken mit festen Datensegmenten sowie in dynamischen Bibliotheken mit verschiebbaren Datensegmenten, die keinen Stack beinhalten, muß die Callback-Funktion die oben angegebene Form aufweisen.

EnumFunc: Der Platzhalter für den anwendungsunterstützten Funktionsnamen.

hWnd: Bezeichnet das Handle zu dem Fenster, das die Eigenschaftsliste enthält.

lpString: Zeigt auf den nullterminierten String, der dem Daten-Handle zugeordnet wurde, nachdem die Anwendung die Funktion *SetProp* zur Festlegung der Eigenschaften aufgerufen hat. Übergibt die Anwendung statt eines Strings ein Atom an die Funktion *SetProp*, so enthält der Parameter *lpString* das Atom im niederwertigen und eine 0 im höherwertigen Word.

hData: Bezeichnet das Daten-Handle.

Rückgabewert Die Callback-Funktion kann jede gewünschte Aufgabe ausführen. Soll die Auflistung fortgesetzt werden, muß der Rückgabewert *True* sein. Ist er *False*, wird die Auflistung beendet.

EnumTaskWindows

Syntax
```
funktion EnumTaskWindows(Task: THandle; EnumFunc:
TFarProc; lParam: LongInt): Bool;
```

Beschreibung Diese Funktion listet alle Fenster auf, die durch den Parameter der Funktion *GetCurrentTask* bezeichnet werden.

Eine Task ist jedes Programm, das als eigenständige Einheit abläuft. Alle Anwendungen werden als Tasks ausgeführt. Jede Instanz einer Anwendung ist wiederum eine Task. Die Auflistung wird dann beendet, wenn die durch den Zeiger *EnumFunc* bezeichnete Callback-Funktion den Wert *False* zurückgibt.

Task: Bezeichnet die Task. Die Funktion *GetCurrentTask* gibt dieses Handle zurück.

EnumFunc: Ist die Adresse der Prozedurinstanz der Callback-Funktion des Fensters.

lParam: Legt den 32-Bit-Wert fest, der zusätzliche Parameter an die durch *EnumFunc* bezeichnete Callback-Funktion übergibt.

Rückgabewert Der Rückgabewert ist *True*, wenn alle zu einer Task gehörenden Fenster aufgelistet wurden, in jedem anderen Fall ist der Rückgabewert *False*.

Callback-Funktion
```
function EnumFunc (Wnd: HWnd; LParam: LongInt): Bool;
export;
```

Beschreibung *EnumFunc:* Platzhalter für den anwendungsunterstützten Funktionsnamen.

hWnd: Bezeichnet ein zur aktuellen Task gehörendes Fenster.

lParam: Bestimmt dasselbe Argument, das an die Funktion *EnumTaskWindows* übergeben wurde.

Rückgabewert Die Callback-Funktion kann jede gewünschte Aufgabe ausführen. Soll die Auflistung fortgesetzt werden, muß der Rückgabewert *True* sein. Ist er *False*, wird die Auflistung beendet.

Querverweis *GetCurrentTask*

EnumWindows

Syntax
```
function EnumWindows(EnumFunc: TFarProc; lParam:
LongInt): Bool;
```

Beschreibung Diese Funktion listet alle übergeordneten Fenster auf dem Bildschirm durch Übergabe der zugehörigen Handles dieser Fenster auf. Die Übergabe erfolgt an die durch den Parameter *EnumFunc* bezeichnete Callback-Funktion. Untergeordnete Fenster werden nicht gelistet. *EnumWindows* fährt mit der Auflistung fort, bis die aufgerufene Funktion den Wert *False* zurückgibt oder das letzte Fenster gelistet wurde.

EnumFunc: Ist die Adresse der Prozedurinstanz der *Callback-Funktion*. Einzelheiten hierzu finden Sie im folgenden Hinweis.

lParam: Legt den Übergabewert an die Callback-Funktion für die Anwendung fest.

Rückgabewert Der Rückgabewert ist *True*, wenn alle Fenster gelistet wurden, andernfalls ist er *False*.

Hinweis Die Adresse, die durch *EnumFunc* übergeben wird, muß durch die Funktion *MakeProcInstance* erzeugt werden.

Callback-Funktion
```
function EnumFunc(Wnd: HWnd; lParam: LongInt) Bool;
export;
```

Beschreibung *EnumFunc:* Der Platzhalter für den anwendungsunterstützten Funktionsnamen.

hWnd: Bezeichnet das Handle des Fensters.

lParam: Legt das 32-Bit-Argument der Funktion *EnumWindows* fest.

Rückgabewert Der Rückgabewert muß *True* sein, wenn die Auflistung fortgesetzt werden soll. *False* stoppt die Auflistung.

EqualRect

Syntax
```
function EqualRect(var Rect1, Rect2: TRect): Bool;
```

Beschreibung Diese Funktion überprüft die Übereinstimmung zweier Rechtecke, indem sie die Koordinaten der oberen linken und der

unteren rechten Ecken vergleicht. Sind die Werte identisch, wird *True* zurückgegeben, andernfalls *False*.

Rect1: Eine *TRect*-Datenstruktur, die die Koordinaten der oberen linken und der unteren rechten Ecke des ersten Rechtecks enthält.

Rect2: Eine *TRect*-Datenstruktur, die die Koordinaten der oberen linken und der unteren rechten Ecke des zweiten Rechtecks enthält.

Rückgabewert Wenn die Rechtecke identisch sind, ist der Rückgabewert *True*, andernfalls ist er *False*.

EqualRgn

Syntax
```
function EqualRgn(SrcRgn1, SrcRgn2: HRgn): Bool;
```

Beschreibung Diese Funktion überprüft zwei vorgegebene Regionen auf ihre Übereinstimmung.

SrcRgn1: Bezeichnet die erste Region.

SrcRgn2: Bezeichnet die zweite Region.

Rückgabewert Der Rückgabewert ist *True*, wenn die Regionen übereinstimmen, andernfalls ist er *False*.

Escape

Syntax
```
function Escape(DC: HDC; Escape, Count: Integer;
InData, OutData: Pointer): Integer;
```

Beschreibung Diese Funktion erlaubt es der Anwendung, auf die Fähigkeiten einer bestimmten Geräteeinheit zuzugreifen, die über die grafische Geräteschnittstelle (GDI) nicht direkt verfügbar sind. *Escape*-Aufrufe einer Anwendung werden übersetzt und an den Gerätetreiber übergeben.

DC: Legt den Gerätekontext fest.

Escape: Bestimmt den Inhalt der *Escape*-Funktion.

Count: Bestimmt die Anzahl der Datenbytes, auf die der Parameter *InData* zeigt.

InData: Zeigt auf die von dieser *Escape*-Funktion benötigte Struktur mit den Eingabedaten.

OutData: Zeigt auf die Datenstruktur und ist zum Empfang der Ausgabe dieser *Escape*-Funktion bestimmt. Wenn keine Daten zurückgegeben werden, muß der Parameter *OutData* den Wert 0 haben.

Rückgabewert Der Rückgabewert ist positiv, wenn die Funktion erfolgreich ausgeführt wurde. Ausnahme ist die *Escape*-Funktion *QueryEscSupport*, die nur die Implementierung überprüft. Der Rückgabewert ist 0, wenn die *Escape*-Funktion nicht implementiert wurde, ein negativer Wert zeigt einen Fehler an.

Die folgende Liste gibt einen Überblick über häufige Fehlermeldungen (vergleichen Sie hierzu Kapitel 1 »Windows-Stile und -Konstanten« im Abschnitt »sp_Xxxx«):

Wert	Bedeutung
sp_Error	Allgemeiner Fehler.
sp_OutOfDisk	Speicherplatz auf der Diskette bzw. Platte ist nicht ausreichend für Spooling. Zusätzlicher Speicherplatz kann nicht verfügbar gemacht werden.
sp_OutOfMemory	Arbeitsspeicher ist für Spooling nicht ausreichend.
sp_UserAbort	Der Benutzer hat den Druckauftrag mit Hilfe des Druck-Managers abgebrochen.

EscapeCommFunction

Syntax
```
function EscapeCommFunction(Cid, Func: Integer):
Integer;
```

Beschreibung Diese Funktion weist die durch den Parameter *Cid* festgelegte serielle Schnittstelle an, die durch den Parameter *Func* bezeichnete erweiterte Funktion auszuführen.

Cid: Legt die serielle Schnittstelle für die Ausführung der erweiterten Funktion fest. Die Funktion *OpenComm* gibt diesen Wert zurück.

Func: Legt den Funktionscode der erweiterten Funktion fest. Folgende Werte stehen zur Verfügung:

Wert	Beschreibung
ClrDTR	Löscht das Data-Terminal-Ready-Signal (DTR).
ClrRTS	Löscht das Request-To-Sent-Signal (RTS).
ResetDev	Wenn möglich, wird das Gerät in den Ausgangszustand zurückgesetzt.
SetDTR	Sendet das Data-Terminal-Ready-Signal.
SetRTS	Sendet das Request-To-Send-Signal.
SetXOff	Beeinflußt die Übertragung so, als wäre das Zeichen *Xoff* empfangen worden.
SetXOn	Beeinflußt die Übertragung so, als wäre das Zeichen *Xon* empfangen worden.

Rückgabewert Der Rückgabewert ist nach erfolgreicher Ausführung der Funktion 0.

Wurde durch den Parameter *Func* ein ungültiger Funktionscode festgelegt, ist der Rückgabewert negativ.

Querverweis *OpenComm*

ExcludeClipRect

Syntax
```
function ExcludeClipRect(DC: HDC; X1, Y1, X2, Y2:
Integer): Integer;
```

Beschreibung Diese Funktion erzeugt eine neue Clipping-Region, die aus der existierenden Clipping-Region abzüglich des festgelegten Rechtecks besteht.

DC: Bezeichnet den Gerätekontext.

X1: Bestimmt die virtuelle x-Koordinate der oberen linken Ecke des Rechtecks.

Y1: Bestimmt die virtuelle y-Koordinate der oberen linken Ecke des Rechtecks.

X2: Bestimmt die virtuelle x-Koordinate der unteren rechten Ecke des Rechtecks.

Y2: Bestimmt die virtuelle y-Koordinate der unteren rechten Ecke des Rechtecks.

Rückgabewert Der Rückgabewert gibt den Typ der neuen Clipping-Region an. Folgende Werte können dabei auftreten:

Wert	Bedeutung
ComplexRegion	Die Region hat überlappende Grenzen.
Error	Es wurde keine Region erzeugt.
NullRegion	Die Region ist leer.
SimpleRegion	Die Region hat keine überlappenden Grenzen.

Hinweis Die durch den absoluten Wert der Differenz *X2 - X1* festgelegte Breite des Rechtecks darf 32 767 Einheiten nicht überschreiten. Diese Grenze gilt auch für die Höhe des Rechtecks (*Y2 - Y1*).

ExcludeUpdateRgn

Syntax
```
function ExcludeUpdateRgn(DC: HDC; Wnd: HWnd): Integer;
```

Beschreibung Diese Funktion verhindert das Zeichnen innerhalb ungültiger Bereiche eines Fensters durch den Ausschluß einer aktualisierten Region im Fenster aus der Clipping-Region.

DC: Bezeichnet den zur Clipping-Region gehörenden Gerätekontext.

Wnd: Bezeichnet das zu aktualisierende Fenster.

Rückgabewert Dieser Wert bestimmt den Typ der resultierenden Region. Folgende Werte sind möglich:

Wert	Bedeutung
ComplexRegion	Die Region hat überlappende Grenzen.
Error	Es wurde keine Region erzeugt.
NullRegion	Die Region ist leer.
SimpleRegion	Die Region hat keine überlappende Grenzen.

ExitWindows

Syntax
```
function ExitWindows(Reserved: LongInt; ReturnCode: Word): Bool;
```

Beschreibung Diese Funktion startet die Standardschließprozedur von Windows. Geben die Anwendungen keine Warnungen zurück, wird die Windows-Sitzung beendet und zu DOS zurückgekehrt. Windows übergibt die Botschaft *wm_QueryEndSession* an die Anwendungen, um anzuzeigen, daß Windows beendet werden soll. Stimmen alle Anwendungen zu, sendet Windows die Botschaft *wm_EndSession* an alle Anwendungen, bevor die Arbeitssitzung endgültig beendet wird.

Reserved: Ist reserviert und sollte auf 0 gesetzt werden.

ReturnCode: Legt den Rückgabewert an DOS nach dem Abschluß von Windows fest.

Rückgabewert Der Rückgabewert ist *False*, wenn eine oder mehrere Anwendungen das Schließen ablehnen. Stimmen alle Anwendungen zu, kehrt die Funktion nicht zurück.

Querverweis *wm_QueryEndSession, wm_EndSession*

ExtFloodFill

Syntax
```
function ExtFloodFill(DC: HDC; X, Y: Integer; Color: TColorRef; FillType: Word): Bool;
```

Beschreibung Diese Funktion füllt einen Bereich der Bildschirmoberfläche mit dem aktuellen Pinsel.

Wird *FillType* auf *FloodFillBorder* gesetzt, geht die Funktion davon aus, daß der gesamte angegebene Bereich von der durch den Parameter *Color* festgelegten Farbe umschlossen ist. Die Funktion beginnt bei dem durch die Parameter *X* und *Y* festgelegten Punkt und füllt das Gebiet in alle Richtungen bis zur angegebenen Farbgrenze.

Wird *FillType* auf *FloodFillSurface* gesetzt, füllt die Funktion

ExtFloodFill ausgehend von dem durch *X* und *Y* festgelegten Punkt alle angrenzenden Gebiete, welche die in *Color* festgelegte Farbe enthalten.

DC: Bezeichnet den Gerätekontext.

X: Bestimmt die virtuelle x-Koordinate des Ausgangspunktes.

Y: Bestimmt die virtuelle y-Koordinate des Ausgangspunktes.

Color: Bestimmt die Farbe der Begrenzung bzw. die des auszufüllenden Bereichs. Die Interpretation dieses Werts hängt vom Parameter *FillType* ab.

FillType: Bestimmt die Art des Ausmalens. Folgende Werte sind möglich:

Wert	Bedeutung
FloodFillBorder	Der auszumalende Bereich wird von der durch *Color* festgelegten Farbe begrenzt. Diese Art des Ausmalens entspricht der der Funktion *FloodFill*.
FloodFillSurface	Der auszumalende Bereich wird durch die in *Color* festgelegte Farbe definiert. Das Ausmalen wird nach außen in allen Richtungen solange fortgesetzt, wie die festgelegte Farbe angetroffen wird. Diese Art des Ausmalens ist insbesondere bei mehrfarbigen Rändern hilfreich.

Rückgabewert

Der Rückgabewert ist bei erfolgreicher Ausführung der Funktion ungleich 0. In folgenden Fällen ist der Rückgabewert 0:

- Das Ausmalen konnte nicht abgeschlossen werden.
- Der vorgegebene Punkt hat die durch *Color* festgelegte begrenzende Farbe (wenn *FloodFillBorder* angefordert wurde).
- Der vorgegebene Punkt hat nicht die durch *Color* festgelegte Farbe (wenn *FloodFillSurface* gesetzt wurde).
- Der Punkt liegt außerhalb der Clipping-Region.

Hinweis

Die Funktion *ExtFloodFill* wird nur von Gerätekontexten im Arbeitsspeicher und von Geräten mit Rasterdarstellungstechnik unterstützt.

ExtTextOut

Syntax

```
function ExtTextOut(DC: HDC; X, Y: Integer; Options:
Word; Rect: PRect; Str: PChar; Count: Word; Dx:
PInteger): Bool;
```

Beschreibung

Diese Funktion schreibt einen String in der aktuellen Schrift in einen rechteckigen Bereich des Bildschirms. Bei dem Recht-

eck kann es sich um einen undurchsichtigen Bereich (ausgefüllt mit der aktuellen Hintergrundfarbe) oder um eine Clipping-Region handeln.

DC: Bezeichnet den Gerätekontext.

X: Bestimmt die virtuelle x-Koordinate für den Ursprung des ersten Zeichens des Strings.

Y: Bestimmt die virtuelle y-Koordinate für den Ursprung des ersten Zeichens des Strings.

Options: Bestimmt den Typ des Rechtecks. Der Parameter kann keinen, einen oder beide der folgenden Werte annehmen: *eto_Clipped* und *eto_Opaque*. Der Wert *eto_Clipped* weist Windows an, den Text an den Rändern des Rechtecks abzuschneiden. Der Wert *eto_Opaque* führt zur Ausfüllung des Rechtecks mit der aktuellen Hintergrundfarbe.

Rect: Zeigt auf eine Datenstruktur vom Typ *TRect*. Der Parameter *Rect* kann ein **nil**-Zeiger sein.

Str: Zeigt auf den angegebenen String.

Count: Gibt die Anzahl der Zeichen in dem String an.

Dx: Zeigt auf ein Array von Werten, welche den Abstand zwischen den Ursprüngen benachbarter Buchstaben festlegen. Zum Beispiel trennen *DX[i]* virtuelle Einheiten die Zeichen i und i + 1.

Rückgabewert Der Rückgabewert zeigt an, ob der String dargestellt wurde. Ist dies der Fall, so ist der Wert *True*, andernfalls ist der Wert *False*.

Hinweis Ist *DX* **nil**, wird der Standardzeichenabstand verwendet.

Die Ursprünge der Zeichenzellen und die Inhalte des Arrays, auf das der Parameter *DX* zeigt, werden in virtuellen Einheiten angegeben. Der Ursprung einer Zeichenzelle ist als ihre obere linke Ecke definiert.

Per Voreinstellung wird die aktuelle Position durch diese Funktion weder benutzt noch aktualisiert. Möglich ist jedoch der Aufruf der Funktion *SetTextAlign* durch die Anwendung. Ist dabei der Parameter *Flags* auf *ta_UpdateCP* gesetzt, wird es Windows ermöglicht, die aktuelle Position jederzeit zu verwenden und zu aktualisieren, wenn die Anwendung die Funktion *ExtTextOut* für einen bestimmten Gerätekontext aufruft. Ist dieses Flag gesetzt, werden die *X*- und *Y*-Koordinaten nachfolgender Aufrufe der Funktion *ExtTextOut* von Windows ignoriert.

Querverweis *SetTextAlign*

FatalAppExit

Syntax
```
procedure FatalAppExit (Action: Word; MessageText PChar);
```

Beschreibung Diese Prozedur stellt ein Meldungsfenster mit dem im Parameter *MessageText* festgelegten Text dar. Die Anwendung wird beendet, wenn das Meldungsfenster geschlossen wird. Wird die Funktion in der Debugging-Version von Windows aufgerufen, kann der Anwender im Meldungsfenster auswählen, ob die Anwendung beendet werden oder zum Aufrufer zurückkehren soll.

Action: Reserviert, muß auf 0 gesetzt sein.

MessageText: Zeigt auf den im Meldungsfenster dargestellten nullterminierten String. Die Meldung wird in einer einzelnen Zeile dargestellt.

Um die Kompatibilität mit Bildschirmen geringer Auflösung zu gewährleisten, sollte der String nicht länger als 35 Zeichen sein.

Hinweis Trifft eine Anwendung auf einen unerwarteten Fehler, sollte sie unter Freigabe ihres gesamten Speichers aus ihrer Botschaftsschleife zurückkehren. Sie sollte *FatalAppExit* nur dann aufrufen, wenn die Anwendung nicht anders beendet werden kann. *FatalAppExit* kann nicht in jedem Fall den gesamten Speicher einer Anwendung freigeben und alle zugehörigen Dateien schließen, dies kann zu einem schweren Windows-Fehler führen.

FatalExit

Syntax
```
procedure FatalExit(Code: Integer);
```

Beschreibung Diese Prozedur stellt den aktuellen Status von Windows auf dem Debugging-Bildschirm dar und fragt Anweisungen zur weiteren Bearbeitung ab.

Es wird der Fehlercode *Code* ausgegeben, gefolgt von einer symbolischen Stack-Darstellung, die die Ausführung bis zu diesem Punkt andeutet.

Anwendungen sollten diese Prozedur nur zum Testen aufrufen. Der Aufruf in der normalen Version von Windows würde zum Abbruch der Anwendung führen.

Code: Bestimmt den darzustellenden Fehlercode.

Hinweis Die Prozedur fordert den Anwender auf, eine Abfrage *Abbrechen*, *Unterbrechen* oder *Ignorieren* zu beantworten.

Die Antwort wird wie folgt bearbeitet:

Antwort	Beschreibung
A	»Abbrechen« – beendet Windows.
U	»Unterbrechen« – simuliert einen nicht-maskierbaren Interrupt (NMI) für den Eintritt in den Debugger.
I	»Ignorieren« – kehrt zum Aufrufer zurück.

Die Prozedur *FatalExit* dient nur zum Testen.

Anwendungen sollten diese Prozedur immer aufrufen, wenn sie einen schweren Fehler feststellen. Alle Ein- und Ausgaben werden über die serielle Standardschnittstelle (AUX) des Computers empfangen bzw. übertragen. Ist ein Debugger installiert, werden sie von diesem verarbeitet.

FillRect

Syntax

```
function FillRect(DC: HDC; var Rect: TRect; Brush:
HBrush): Integer;
```

Beschreibung Diese Funktion füllt ein gegebenes Rechteck mit dem angegebenen Pinsel aus, wobei das gesamte Rechteck einschließlich des linken und oberen Randes, jedoch ohne den rechten und unteren Rand, ausgefüllt wird.

DC: Bezeichnet den Gerätekontext.

Rect: Eine *TRect*-Datenstruktur, die die virtuellen Koordinaten des auszufüllenden Rechtecks enthält.

Brush: Bezeichnet den Pinsel, der beim Ausfüllen des Rechtecks benutzt wird.

Rückgabewert Obwohl der Rückgabewert vom Typ Integer ist, wird er nicht benutzt und hat keine Bedeutung.

Hinweis Der Pinsel muß vorher mit einer der folgenden Funktionen erstellt worden sein: *CreateHatchBrush*, *CreatePatternBrush* oder *CreateSolidBrush*. Er kann auch mit der Funktion *GetStockObject* abgerufen werden.

Beim Ausfüllen des angegebenen Rechtecks schließt die Funktion *FillRect* die rechte und untere Seite des Rechtecks nicht ein. Die GDI füllt ein Rechteck ungeachtet des Abbildungsmodus bis zur rechten Spalte und der untersten Zeile aus, diese sind aber nicht eingeschlossen.

FillRect vergleicht die Werte der Felder *top*, *bottom*, *left* und *right* des bezeichneten Rechtecks. Wenn *bottom* kleiner oder gleich *top* ist bzw. *right* kleiner oder gleich *left* ist, wird das Rechteck nicht gezeichnet.

Querverweis *CreateHatchBrush, CreatePatternBrush, CreateSolidBrush, GetStockObject*

FillRgn

Syntax
```
function FillRgn(DC: HDC; Rgn: HRgn; Brush: HBrush): Bool;
```

Beschreibung Diese Funktion füllt die durch den Parameter *Rgn* angegebene Region mit dem durch den Parameter *Brush* bezeichneten Pinsel.

DC: Bezeichnet den Gerätekontext.

Rgn: Bezeichnet die Region, die ausgefüllt werden soll. Die Koordinaten für die vorgegebene Region sind in gerätebezogenen Einheiten angegeben.

Brush: Bezeichnet den Pinsel, der benutzt werden soll, um die Region auszufüllen.

Rückgabewert Der Rückgabewert ist *True*, wenn die Funktion erfolgreich war, andernfalls ist er *False*.

FindAtom

Syntax
```
function FindAtom(Str: PChar): TAtom;
```

Beschreibung Diese Funktion durchsucht die Atomtabelle nach dem String, auf den der Parameter *Str* zeigt, und ruft das dem String zugeordnete Atom ab.

Str: Zeigt auf den nullterminierten String, nach dem gesucht werden soll.

Rückgabewert Der Rückgabewert bezeichnet das Atom, das dem String zugeordnet ist. Er ist 0, wenn der String nicht in der Tabelle ist.

FindResource

Syntax
```
function FindResource(Instance: THandle; Name, ResType: PChar): THandle;
```

Beschreibung Diese Funktion bestimmt die Position einer Ressource in der angegebenen Ressourcendatei. Die Parameter *Name* und *ResType* geben Namen und Typ der Ressource an.

Instance: Bezeichnet die Instanz des Moduls, dessen ausführbare Datei die Ressource enthält.

Name: Zeigt auf einen nullterminierten String, der den Namen der Ressource darstellt.

ResType: Zeigt auf einen nullterminierten String, der die Typbezeichnung der Ressource darstellt. Für vordefinierte Ressour-

centypen sollte der Parameter *ResType* einen der folgenden Werte annehmen:

Wert	Bedeutung
rt_Accelerator	Accelerator-Tabelle.
rt_Bitmap	Bitmap-Ressource.
rt_Cursor	Mauszeigerressource.
rt_Dialog	Dialogfenstervorlage.
rt_Font	Schriftressource.
rt_FontDir	Schriftenverzeichnisressource.
rt_Icon	Symbolressource.
rt_Menu	Menüressource.
rt_RcData	Benutzerdefinierte Ressource (Rohdaten).
rt_String	String-Ressource.

Rückgabewert Bei erfolgreicher Ausführung der Funktion bezeichnet der Rückgabewert die benannte Ressource. Er ist 0, wenn die angeforderte Ressource nicht gefunden werden kann.

Hinweis Eine Anwendung darf die Funktionen *FindResource* und *LoadResource* nicht aufrufen, um Mauszeiger-, Symbol- und Stringressourcen zu laden. Dies geschieht durch Aufruf einer der folgenden Funktionen:

LoadCursor
LoadIcon
LoadString

Eine Anwendung kann die Funktionen *FindResource* und *LoadResource* aufrufen, um andere vordefinierte Ressourcentypen zu laden. Es wird jedoch empfohlen, die entsprechenden Ressourcen durch Aufrufen einer der folgenden Funktionen zu laden:

LoadAccelerators
LoadBitmap
LoadMenu

Ist das höherwertige Word der Parameter *Name* oder *ResType* 0, gibt das niederwertige Word die Integer-ID des Namens oder des Typs der angegebenen Ressource an. Andernfalls sind die Parameter Zeiger auf nullterminierte Strings.

Wenn das erste Zeichen des Strings ein Nummernzeichen ist (#), stehen die übrigen Zeichen für eine Dezimalzahl, die die Integer-ID des Ressourcennamens oder des Ressourcentyps angibt. So bedeutet zum Beispiel der String #258 die Integer-ID 258.

Um den Speicherplatzbedarf der von der Anwendung benutzten Ressourcen zu senken, sollten sie sich mittels Integer-ID auf die Ressourcen beziehen, nicht über den Namen.

FindWindow

Syntax
```
function FindWindow(ClassName, WindowName: PChar): HWnd;
```

Beschreibung Diese Funktion gibt das Handle des Fensters zurück, dessen Klasse mit dem Parameter *ClassName* angegeben ist und dessen Überschrift durch *WindowName* angegeben wird.

ClassName: Zeigt auf einen nullterminierten String, der die Klassenbezeichnung des Fensters angibt. Ist *ClassName* 0, gelten alle Klassenbezeichnungen.

WindowName: Zeigt auf einen nullterminierten String, der den Fensternamen angibt (dem Text in der Titelleiste des Fensters). Ist *WindowName* 0, gelten alle Fensternamen.

Rückgabewert Der Rückgabewert bezeichnet das Fenster, das den angegebenen Klassennamen und Fensternamen hat. Er ist 0, wenn kein solches Fenster gefunden wurde.

FlashWindow

Syntax
```
function FlashWindow(Wnd: HWnd; Invert: Bool): Bool;
```

Beschreibung Diese Funktion läßt ein angegebenes Fenster einmal aufblinken, d.h. das Erscheinungsbild seiner Titelleiste wird so verändert, als ob das Fenster vom inaktiven in den aktiven Zustand übergegangen wäre oder umgekehrt.

Normalerweise läßt man ein Fenster aufblinken, um den Benutzer darüber zu informieren, daß das Fenster Aufmerksamkeit verdient, aber den Fokus gegenwärtig nicht hat.

Wnd: Bezeichnet das Fenster, das aufblinken soll. Das Fenster kann entweder offen oder ein Symbol sein.

Invert: Bestimmt, ob das Fenster aufleuchten oder wieder in seinen ursprünglichen Zustand versetzt werden soll. Ist *Invert True*, wird das Fenster von einem Zustand in den anderen versetzt, ist er *False*, wird das Fenster wieder in seinen ursprünglichen Zustand versetzt (entweder aktiv oder inaktiv).

Rückgabewert Der Rückgabewert gibt den Status des Fensters vor dem Aufruf von *FlashWindow* an. Er ist *True*, wenn das Fenster vor dem Aufruf aktiv war, andernfalls *False*.

Hinweis Die Funktion läßt das Fenster nur einmal aufblinken, für fortwährendes Blinken sollte die Funktion einen System-Timer erstellen.

Der Parameter *Invert* sollte nur *False* sein, wenn das Fenster den Fokus erhält und nicht mehr aufblinken soll; er sollte bei aufeinanderfolgenden Aufrufen *True* sein, solange das Fenster darauf wartet, den Fokus zu erhalten.

Diese Funktion gibt für Symbole immer *True* zurück. Wenn das Fenster als Symbol existiert, läßt *FlashWindow* nur das Symbol aufleuchten; für Fenstersymbole wird *Invert* ignoriert.

FloodFill

Syntax
```
function FloodFill(DC: HDC; X, Y: Integer; Color:
TColorRef): Bool;
```

Beschreibung Diese Funktion füllt einen Bereich auf der Bildschirmfläche mit dem aktuellen Pinsel aus. Sie geht davon aus, daß der Bereich gemäß dem vom Parameter *Color* angegebenen Wert eingegrenzt wird. Die Funktion beginnt an dem durch die Parameter *X* und *Y* bezeichneten Punkt und fährt dann in alle Richtungen bis zur Farbbegrenzung fort.

DC: Bezeichnet den Gerätekontext.

X: Bestimmt die virtuelle x-Koordinate des Punktes, von dem aus das Ausfüllen durchgeführt wird.

Y: Bestimmt die virtuelle y-Koordinate des Punktes, von dem aus das Ausfüllen durchgeführt wird.

Color: Bestimmt die Grenzfarbe.

Rückgabewert Der Rückgabewert ist *True*, wenn die Funktion erfolgreich ausgeführt wurde. Er ist *False*, wenn das Ausfüllen nicht zu Ende geführt werden konnte, der gegebene Punkt selbst die von *Color* bezeichnete Farbe hat oder der Punkt außerhalb der Clipping-Region liegt.

Hinweis Die Funktion *FloodFill* kann nur bei Speichergerätekontexten und Geräten, die Rasterdarstellungstechnik unterstützen, verwendet werden. Weitere Informationen finden Sie unter der Beschreibung der Funktion *GetDeviceCaps*.

FlushComm

Syntax
```
function FlushComm(Cid, Queue: Integer): Integer;
```

Beschreibung Diese Funktion leert die Sende- oder Empfangs-Warteschlange der durch den Parameter *Cid* angegeben Schnittstelle. Der Parameter *Queue* bestimmt, welche Warteschlange geleert werden soll.

Cid: Bestimmt die Schnittstelle, deren Warteschlange geleert werden soll. Die Funktion *OpenComm* gibt diesen Wert zurück.

Queue: Bestimmt die Warteschlange, die geleert werden soll. Ist *Queue* 0, wird die Sende-Warteschlange geleert, ist er 1, wird die Empfangs-Warteschlange geleert.

Rückgabewert Der Rückgabewert ist 0, wenn die Funktion erfolgreich ausgeführt wurde. Er ist negativ, wenn *Cid* ein ungültiges Gerät oder *Queue* eine ungültige Warteschlange bezeichnet.

Querverweis *OpenComm*

FrameRect

Syntax
```
procedure FrameRect(DC: HDC; var Rect: TRect; Brush:
HBrush);
```

Beschreibung Diese Prozedur zeichnet einen Rahmen um das durch den Parameter *Rect* bezeichnete Rechteck und verwendet dabei den gegebenen Pinsel. Die Breite und Höhe des Rahmens beträgt immer eine virtuelle Einheit.

DC: Bezeichnet den Gerätekontext des Fensters.

Rect: Eine *TRect*-Datenstruktur, die die virtuellen Koordinaten der oberen linken und der unteren rechten Ecke des Rechtecks enthält.

Brush: Bezeichnet den Pinsel, der zum Einrahmen des Rechtecks verwendet werden soll.

Hinweis Der von *Brush* bezeichnete Pinsel muß vorher mit einer der Funktionen *CreateHatchBrush, CreatePatternBrush* oder *CreateSolidBrush* erstellt worden sein.

Wenn *bottom* kleiner oder gleich *top* ist bzw. *right* kleiner oder gleich *left* ist, wird der Rahmen des Rechtecks nicht gezeichnet.

Querverweis *CreateHatchBrush, CreatePatternBrush, CreateSolidBrush*

FrameRgn

Syntax
```
function FrameRgn(DC: HDC; Rgn: HRgn; Brush: HBrush;
Width, Height: Integer): Bool;
```

Beschreibung Diese Funktion zeichnet mit dem durch den Parameter *Brush* bezeichneten Pinsel einen Rahmen um die die durch den Parameter *Rgn* vorgegebene Region. Der Parameter *Width* gibt die Breite des Rahmens in vertikalen, der Parameter *Height* die Höhe des Rahmens in horizontalen Pinselstrichen an.

DC: Bezeichnet den Gerätekontext.

Rgn: Bezeichnet die Region, die in einen Rahmen eingeschlossen werden soll. Die Koordinaten für die angegebene Region sind in gerätebezogenen Einheiten gegeben.

Brush: Bezeichnet den Pinsel, der benutzt werden soll, um den Rahmen zu zeichnen.

Width: Bestimmt die Breite in vertikalen Pinselstrichen (in virtuellen Einheiten).

Height: Bestimmt die Höhe in horizontalen Pinselstrichen (in virtuellen Einheiten).

Rückgabewert Der Rückgabewert ist *True*, wenn die Funktion erfolgreich war, andernfalls ist er *False*.

FreeLibrary

Syntax
```
procedure FreeLibrary(LibModule: THandle);
```

Beschreibung Diese Prozedur verringert den Referenzzähler des geladenen Bibliothekmoduls um 1.

Wenn der Referenzzähler 0 erreicht, wird der vom Modul belegte Speicherplatz freigegeben.

LibModule: Bezeichnet das geladene Bibliotheksmodul.

FreeModule

Syntax
```
function FreeModule(Module: THandle): Bool;
```

Beschreibung Diese Funktion verringert den Referenzzähler des geladenen Moduls um 1. Wenn der Referenzzähler 0 erreicht, wird der vom Modul belegte Speicherplatz freigegeben.

Module: Bezeichnet das geladene Modul.

Rückgabewert Wird nicht verwendet.

FreeProcInstance

Syntax
```
procedure FreeProcInstance(Proc: TFarProc);
```

Beschreibung Die durch den Parameter *Proc* bezeichnete Funktion wird durch *FreeProcInstance* aus dem Datensegment, mit dem sie durch die Funktion *MakeProcInstance* verknüpft wurde, befreit.

Proc: Ist die Adresse der Prozedurinstanz der freizugebenden Funktion. Sie muß vorher mit der Funktion *MakeProcInstance* erstellt worden sein.

Hinweis Nach dem Freigeben einer Prozedurinstanz führen sämtliche Versuche, die Funktion mittels der freigegebenen Adresse der Prozedurinstanz aufzurufen, zu einem nicht behebbaren Fehler.

Querverweis *MakeProcInstance*

FreeResource

Syntax
```
function FreeResource(ResData: THandle): Bool;
```

Beschreibung Diese Funktion löscht eine geladene Ressource aus dem Speicher, indem sie den reservierten Speicherplatz, der von der Ressource belegt war, freigibt.

FreeResource gibt die Ressource erst dann tatsächlich frei, wenn der Referenzzähler 0 ist. Dies ist der Fall, wenn *FreeResource* für diese Ressource genauso oft aufgerufen wurde wie *LoadResource*. Dadurch wird sichergestellt, daß die Daten solange im Speicher verbleiben, wie sie von der Anwendung benötigt werden.

ResData: Bezeichnet die Daten, die der Ressource zugeordnet sind. Es wird unterstellt, daß das Handle mit der Funktion *LoadResource* erzeugt wurde.

Rückgabewert Bei erfolgreicher Ausführung der Funktion ist der Rückgabewert *False*, er ist *True*, wenn die Ressource nicht freigegeben wurde.

Querverweis *LoadResource*

FreeSelector

Syntax
```
function FreeSelector(Selector: Word): Word;
```

Beschreibung Diese Funktion gibt einen Selektor frei, der ursprünglich von den Funktionen *AllocSelector* oder *AllocDStoCSAlias* zugewiesen wurde. Nachdem die Anwendung *FreeSelector* aufgerufen hat, ist der Selektor ungültig und darf nicht weiter verwendet werden.

Selector: Bestimmt den Selektor, der freigegeben werden soll.

Rückgabewert Bei erfolgreicher Ausführung der Funktion ist der Rückgabewert 0, andernfalls ist er der mit *Selector* vorgegebene Selektor.

Hinweis Eine Anwendung sollte diese Funktion nur benutzen, wenn dies unbedingt nötig ist. Die Verwendung dieser Funktion entspricht nicht den gängigen Windows-Programmiertechniken.

Querverweis *AllocSelector, AllocDStoCSAlias*

GetActiveWindow

Syntax
```
function GetActiveWindow: HWnd;
```

Beschreibung Diese Funktion liefert das Handle des aktiven Fensters. Das aktive Fenster ist entweder das mit dem aktuellen Fokus oder

das durch die Funktion *SetActiveWindow* explizit aktivierte Fenster.

Die Funktion hat keine Parameter.

Rückgabewert Der Rückgabewert bezeichnet das aktive Fenster.

Querverweis *SetActiveWindow*

GetAspectRatioFilter

Syntax
```
function GetAspectRatioFilter(DC: HDC): LongInt;
```

Beschreibung Diese Funktion ermittelt die aktuellen Einstellungen für das Verhältnis von Bildschirmhöhe und -breite. Dieses Verhältnis wird durch die Breite und Höhe der Pixel eines bestimmten Geräts bestimmt. Informationen über dieses Höhen-/Seitenverhältnis werden für die Herstellung, Auswahl und Darstellung von Schriften benötigt.

Windows stellt einen besonderen Filter für die Auswahl von Schriften zur Verfügung, der die für dieses spezielle Höhen-/Seitenverhältnis geeigneten Schriften aus allen vorhandenen Schriften filtert. Der Filter arbeitet mit dem durch die Funktion *SetMapperFlags* festgelegten Verhältnis.

DC: Bezeichnet den Gerätekontext, der das Höhen-/Seitenverhältnis enthält.

Rückgabewert Der Rückgabewert bestimmt das Höhen-/Seitenverhältnis des aktuellen Filters. Die x-Koordinate des Höhen-/Seitenverhältnis ist im höherwertigen Word, die y-Koordinate im niederwertigen Word abgelegt.

Querverweis *SetMapperFlags*

GetAsyncKeyState

Syntax
```
function GetAsyncKeyState(Key: Integer): Integer;
```

Beschreibung Diese Funktion überprüft, ob im Moment des Funktionsaufrufs eine Taste gedrückt ist und ob diese Taste seit dem letzten Aufruf von *GetAsyncKeyState* gedrückt wurde. Ist das höchstwertige Bit des Rückgabewerts gesetzt, wird die Taste gerade gedrückt, ist das niederwertigste Bit gesetzt, wurde die Taste seit dem letzten Funktionsaufruf betätigt.

Key: Legt einen der 256 möglichen virtuellen Tastencodes fest.

Rückgabewert Der Rückgabewert gibt an, ob die Taste seit dem letzten Aufruf von *GetAsyncKeyState* gedrückt wurde und ob sie aktuell gedrückt wird. Ist das höchstwertige Bit gesetzt, wird die Taste gerade gedrückt. Ist das niederwertigste Bit gesetzt, wurde die Taste nach dem vorherigen Aufruf dieser Funktion betätigt.

GetAtomHandle

Syntax
```
function GetAtomHandle(AnAtom: TAtom): THandle;
```

Beschreibung Die Funktion liefert ein Handle (relativ zum lokalen Heap) für den String, der zu dem durch *AnAtom* bezeichneten Atom gehört.

AnAtom: Bestimmt den Atom-Bezeichner.

Rückgabewert Der Rückgabewert bezeichnet den zum Atom gehörigen String. Er ist 0, wenn kein solches Atom existiert.

GetAtomName

Syntax
```
function GetAtomName(AnAtom: TAtom; Buffer: PChar;
Size: Integer): Word;
```

Beschreibung Diese Funktion erstellt eine Kopie des zum Parameter *AnAtom* gehörigen Strings in dem Puffer, auf den der Parameter *Buffer* zeigt. *Size* legt die maximale Größe des Puffers fest.

AnAtom: Bezeichnet den aufzufindenden String.

Buffer: Zeigt auf den Puffer, der den String aufnehmen soll.

Size: Bestimmt die maximale Größe des Puffers in Bytes.

Rückgabewert Der Rückgabewert bestimmt die Anzahl von Bytes, die in den Puffer kopiert wurden. Er ist 0, wenn das angegebene Atom ungültig ist.

GetBitmapBits

Syntax
```
function GetBitmapBits(Bitmap: HBitmap; Count: LongInt;
Bits: Pointer): LongInt;
```

Beschreibung Diese Funktion kopiert die Bits des angegebenen Bitmaps in den Puffer, auf den der Parameter *Bits* zeigt. Der Parameter *Count* bestimmt die Anzahl der zu kopierenden Bytes. Zur Feststellung des richtigen Werts von *Count* für das angegebene Bitmap sollte die Funktion *GetObject* verwendet werden.

Bitmap: Bezeichnet das Bitmap.

Count: Legt die Anzahl der zu kopierenden Bytes fest.

Bits: Zeiger auf den Puffer, der das Bitmap speichern soll. Das Bitmap ist ein Array von Bytes, dieses Array entspricht einem horizontalen Raster, das ein Vielfaches von 16 Bit darstellt.

Rückgabewert Der Rückgabewert bestimmt die tatsächliche Anzahl von Bytes in dem Bitmap. Liegt ein Fehler vor, ist er 0.

GetBitmapDimension

Syntax
```
function GetBitmapDimension(Bitmap: HBitmap): LongInt;
```

Beschreibung Diese Funktion gibt die Breite und Höhe des durch den Parameter *Bitmap* bestimmten Bitmaps zurück. Höhe und Breite des Bitmaps sollten zuvor durch die Funktion *SetBitmapDimension* gesetzt werden.

Bitmap: Bezeichnet das Bitmap.

Rückgabewert Der Rückgabewert gibt Breite und Höhe des Bitmaps in zehntel Millimetern an. Der Wert der Höhe liegt im höherwertigen, der der Breite im niederwertigen Word. Wurden diese Werte nicht zuvor durch die Funktion *SetBitmapDimension* gesetzt, ist der Rückgabewert 0.

Querverweis *SetBitmapDimension*

GetBkColor

Syntax
```
function GetBkColor(DC: HDC): LongInt;
```

Beschreibung Die Funktion gibt die aktuelle Hintergrundfarbe des angegebenen Geräts zurück.

DC: Bezeichnet den Gerätekontext.

Rückgabewert Der Rückgabewert gibt den RGB-Farbwert der aktuellen Hintergrundfarbe an.

GetBkMode

Syntax
```
function GetBkMode(DC: HDC): Integer;
```

Beschreibung Diese Funktion gibt den Hintergrundmodus des festgelegten Geräts zurück. Der Hintergrundmodus wird für Texte, Schraffurpinsel und Stiftarten verwendet, die keine durchgehende Linie ziehen.

DC: Bezeichnet den Gerätekontext.

Rückgabewert Der Rückgabewert gibt den aktuellen Hintergrundmodus an. Möglich sind sind die Werte *Opaque* und *Transparent* (siehe Kapitel 1 »Windows-Stile und -Konstanten«.)

GetBrushOrg

Syntax
```
function GetBrushOrg(DC: HDC): LongInt;
```

Beschreibung Diese Funktion ermittelt den aktuellen Ursprung des Pinsels im gegebenen Gerätekontext.

DC: Bezeichnet den Gerätekontext.

Rückgabewert Der Rückgabewert gibt den aktuellen Ursprung des Pinsels an. Die x-Koordinate ist im niederwertigen, die y-Koordinate im höherwertigen Word abgelegt. Die Koordinaten werden in Geräteeinheiten angegeben.

Hinweis Zu Beginn hat der Ursprung des Pinsels die Koordinaten (0,0).

GetBValue

Syntax
```
function GetBValue(RGB: LongInt): Byte;
```

Beschreibung Diese Inline-Funktion ermittelt den blauen Farbanteil eines RGB-Farbwerts.

RGB: Gibt ein rotes, ein grünes und ein blaues Farbfeld an. Diese Felder bestimmen die Intensität der gegebenen Farbe.

Rückgabewert Der Rückgabewert ist ein Byte, das den blauen Farbwert des Parameters *RGB* enthält.

Hinweis Der Wert 255 entspricht der maximalen Intensität eines einzelnen Bytes, 0 entspricht der minimalen Intensität eines einzelnen Bytes.

GetCapture

Syntax
```
function GetCapture: HWnd;
```

Beschreibung Diese Funktion stellt ein Handle bereit, welches das Fenster bezeichnet, das den Mausfang hat. Es kann jeweils nur ein einzelnes Fenster den Mausfang besitzen, dieses empfängt die Mauseingaben auch dann, wenn sich der Mauszeiger außerhalb seiner Grenzen befindet.

Diese Funktion hat keine Parameter.

Rückgabewert Der Rückgabewert bezeichnet das Fenster, das den Mausfang hat. Er ist 0, wenn kein Fenster den Mausfang hat.

Hinweis Ein Fenster erhält den Mausfang, wenn sein Handle als Parameter *Wnd* an die Funktion *SetCapture* übergeben wird.

Querverweis *SetCapture*

GetCaretBlinkTime

Syntax
```
function GetCaretBlinkTime: Word;
```

Beschreibung Diese Funktion liefert die Blinkrate des Carets, die den Zeitraum zwischen dem Aufblinken des Carets in Millisekunden bestimmt. Die Funktion hat keine Parameter.

Rückgabewert Der Rückgabewert gibt die Blinkrate in Millisekunden an.

GetCaretPos

Syntax
```
procedure GetCaretPos(var Point: TPoint);
```

Beschreibung Diese Prozedur ermittelt die aktuelle Position des Carets in Form von Bildschirmkoordinaten und kopiert sie in die *TPoint*-Struktur *Point*.

Point: Eine *TPoint*-Struktur, die die Bildschirmkoordinaten des Carets aufnehmen soll.

Hinweis Die Position des Carets wird immer in den Client-Koordinaten des Fensters angegeben, in dem sich das Caret befindet.

GetCharWidth

Syntax
```
function GetCharWidth(DC: HDC; FirstChar, LastChar:
Word; var Buffer): Bool;
```

Beschreibung Diese Funktion ermittelt die Breite einzelner Zeichen in einer fortlaufenden Gruppe von Zeichen der aktuellen Schrift. Wird zum Beispiel mit dem Parameter *FirstChar* das Zeichen *a* und mit dem Parameter *LastChar* das Zeichen *z* angegeben, ermittelt die Funktion *GetCharWidth* die Breiten aller Kleinbuchstaben. Die Werte werden in dem Puffer abgelegt, auf den *Buffer* zeigt.

DC: Bezeichnet den Gerätekontext.

FirstChar: Gibt das erste Zeichen einer fortlaufenden Gruppe von Zeichen der aktuellen Schrift an.

LastChar: Gibt das letzte Zeichen einer fortlaufenden Gruppe von Zeichen der aktuellen Schrift an.

Buffer: Zeigt auf einen Puffer, der die Werte der Breite einer fortlaufenden Gruppe von Zeichen der aktuellen Schrift aufnimmt.

Rückgabewert Bei erfolgreicher Ausführung der Funktion ist der Rückgabewert *True*, andernfalls *False*.

Hinweis Existiert ein Zeichen der fortlaufenden Gruppe in einer bestimmten Schrift nicht, wird der Wert der Breite des Standardzeichens zugewiesen.

GetClassInfo

Syntax
```
function GetClassInfo(Instance: THandle; ClassInfo:
PChar; var WndClass: TWndClass): Bool;
```

Beschreibung Diese Funktion ermittelt Informationen über eine Fensterklasse. Der Parameter *Instance* bezeichnet die Instanz der Anwendung, welche die Fensterklasse erstellt hat, und der Parameter *ClassInfo* die Fensterklasse.

Findet die Funktion die angegebene Fensterklasse, werden die *TWndClass*-Daten zur Registrierung der Fensterklasse in die durch den Parameter *WndClass* bezeichnete Datenstruktur vom Typ *TWndClass* kopiert.

Instance: Bezeichnet die Instanz der die Fensterklasse erzeugenden Anwendung. Zur Ermittlung von Klassen, die durch Windows definiert wurden (beispielsweise Aktionsschalter oder Listen), muß *Instance* auf 0 gesetzt werden.

ClassInfo: Zeigt auf einen nullterminierten String, der den Namen der gesuchten Klasse enthält.

WndClass: Struktur vom Typ *TWndClass*, in welche die Informationen über die Klasse kopiert werden.

Rückgabewert Der Rückgabewert hat den Wert *True*, wenn die Funktion eine passende Klasse gefunden und die Daten erfolgreich kopiert hat.

Der Rückgabewert ist *False*, wenn keine entsprechende Klasse gefunden wurde.

Hinweis Die Felder *lpszlassName*, *lpszMenuName* und *Instance* aus der *TWndClass*-Datenstruktur werden durch diese Funktion nicht zurückgegeben. Der Menüname ist nicht intern gespeichert und kann daher nicht zurückgegeben werden. Der Klassenname ist bereits bekannt, da er an diese Funktion übergeben wurde. Alle anderen Felder werden mit den Werten aus der Klassenregistrierung zurückgegeben.

GetClassLong

Syntax
```
function GetClassLong(Wnd: HWnd; Index: Integer): LongInt;
```

Beschreibung Diese Funktion ermittelt den durch *Index* festgelegten Wert vom Typ LongInt aus der *TWndClass*-Struktur des durch den Parameter *Wnd* bezeichneten Fensters.

Wnd: Bezeichnet das Fenster.

Index: Bestimmt den Byte-Offset des zu ermittelnden Werts. Auch folgender Wert ist möglich:

Wert	Bedeutung
gcl_WndProc	Bestimmt einen Zeiger auf die Fensterfunktion.

Rückgabewert Der Rückgabewert bestimmt den aus der *TWndClass*-Struktur ermittelten Wert.

Hinweis Um auf einen bei der Erstellung der Fensterklassenstruktur zusätzlich reservierten Vier-Byte-Wert zuzugreifen, muß ein positiver Byte-Offset für den durch *Index* angegebenen Index verwendet werden. Der erste Vier-Byte-Wert im zusätzlich belegten Speicher liegt bei Offset 0, der nächste bei Offset 4, usw.

GetClassName

Syntax
```
function GetClassName(Wnd: HWnd; ClassName: PChar;
MaxCount: Integer): Integer;
```

Beschreibung Diese Funktion ermittelt den Klassennamen des durch den Parameter *Wnd* bezeichneten Fensters.

Wnd: Bezeichnet das Fenster, dessen Klassenname ermittelt werden soll.

ClassName: Zeigt auf einen Puffer, in dem der Klassenname abgelegt wird.

MaxCount: Legt die maximale Anzahl der im Parameter *ClassName* abzulegenden Bytes fest. Ist der Name länger, wird er abgeschnitten.

Rückgabewert Der Rückgabewert gibt die tatsächlich in *ClassName* kopierte Zeichenzahl an. Der Wert ist 0, wenn der Klassenname ungültig ist.

GetClassWord

Syntax
```
function GetClassWord(Wnd: HWnd, Index: Integer): Word;
```

Beschreibung Diese Funktion ermittelt das durch *Index* bezeichnete Word aus der *TWndClass*-Struktur. Das zugrundeliegende Fenster wird durch den Parameter *Wnd* festgelegt.

Wnd: Bezeichnet das Fenster.

Index: Bestimmt den Byte-Offset des zu ermittelnden Wertes. Auch folgende Werte sind möglich:

Wert	Bedeutung
gcw_CBClsExtra	Bestimmt, wieviel Bytes zusätzlicher Klasseninformationen existieren. Informationen zum Zugriff auf diesen Speicher finden sich im folgenden Hinweis.

Wert	Bedeutung
gcw_CBWndExtra	Bestimmt, wieviel Bytes zusätzlicher Fensterinformationen existieren. Informationen zum Zugriff auf diesen Speicher finden sich im folgenden Hinweis.
gcw_HBrBackground	Stellt ein Handle für den Hintergrundpinsel zur Verfügung.
gcw_HCursor	Stellt ein Handle für den Mauszeiger zur Verfügung.
gcw_HICon	Stellt ein Handle für das Symbol zur Verfügung.
gcw_HModule	Stellt ein Handle für das Modul zur Verfügung.
gcw_Style	Ermittelt das Stil-Bit der Fensterklasse.

Rückgabewert Der Rückgabewert gibt den in der *TWndClass*-Struktur gefundenen Wert an.

Hinweis Um auf Zwei-Byte-Werte zuzugreifen, die bei der Erstellung der Fensterklassenstruktur zusätzlich reserviert wurden, muß ein positiver Byte-Offset für den durch den Parameter *Index* angegebenen Index verwendet werden. Der erste Zwei-Byte-Wert liegt im zusätzlich belegten Speicher bei Offset 0, der folgende bei Offset 2, usw.

GetClientRect

Syntax
```
procedure GetClientRect(Wnd: HWnd; var Rect: TRect);
```

Beschreibung Diese Prozedur kopiert die Client-Koordinaten des Client-Bereichs eines Fensters in die *TRect*-Struktur *Rect*. Diese Koordinaten bestimmen die obere linke und untere rechte Ecke des Client-Bereichs. Die Koordinaten werden relativ zur oberen linken Ecke dieses Client-Bereichs, die die Koordinaten (0,0) hat, gebildet.

Wnd: Bezeichnet das dem Client-Bereich zugeordnete Fenster.

Rect: Eine *TRect*-Datenstruktur.

GetClipboardData

Syntax
```
function GetClipboardData(Format: Word): THandle;
```

Beschreibung Diese Funktion ruft Daten aus der Zwischenablage in dem durch den Parameter *Format* angegebenen Format ab. Die Zwischenablage muß vorher geöffnet werden.

Format: Bestimmt ein Datenformat. Die Beschreibung der Datenformate ist in Kapitel 1 »Windows-Stile und -Konstanten« in Abschnitt »cf_Xxxx« nachzulesen.

Rückgabewert Der Rückgabewert bezeichnet den Speicherblock mit den Daten der Zwischenablage. Der Typ des Handles ist von dem durch *Format* festgelegten Datenformat abhängig. Liegt ein Fehler vor, ist der Rückgabewert 0.

Hinweis Die verfügbaren Formate können vorab durch die Funktion *EnumClipboardFormats* aufgelistet werden.

Das von *GetClipboardData* zurückgegebene Daten-Handle wird von der Zwischenablage, nicht von der Anwendung kontrolliert. Um eine längere Abhängigkeit von diesem Daten-Handle zu vermeiden, sollte die Anwendung diese Daten sofort kopieren. Die Anwendung sollte das Daten-Handle weder freigeben noch gesperrt lassen.

Windows unterstützt die Textformate *cf_Text* und *cf_OEMText*. *cf_Text* ist das Standardtextformat der Zwischenablage, *cf_OEMText* wird für das Textformat in Nicht-Windows-Anwendungen verwendet. Wird *GetClipboardData* zum Abrufen von Daten in einem bestimmten Format verwendet und ist nur das andere Format verfügbar, nimmt Windows automatisch vor Übergabe der Daten an die Anwendung die Konvertierung in das gewünschte Format vor.

Enthält die Zwischenablage Daten im Format *cf_Palette* (virtuelle Farbpalette), sollte die Anwendung davon ausgehen, daß alle anderen Daten in der Zwischenablage bezüglich dieser virtuellen Palette realisiert sind.

Querverweis *SetClipboardData*

GetClipboardFormatName

Syntax
```
function GetClipboardFormatName(Format: Word;
FormatName: PChar; MaxCount: Integer): Integer;
```

Beschreibung Diese Funktion ermittelt den Namen des durch den Parameter *Format* bestimmten registrierten Formats aus der Zwischenablage. Der Name wird in den Puffer kopiert, auf den *FormatName* zeigt.

Format: Bestimmt den Typ des zu ermittelnden Formats. Möglich sind die in Kapitel 1 »Windows-Stile und -Konstanten« im Abschnitt »cf_Xxxx« beschriebenen Werte.

FormatName: Zeigt auf den Puffer, der den Namen des Formats aufnehmen soll.

MaxCount: Legt die maximale Länge des in den Puffer zu kopierenden Strings in Bytes fest.

Ist der Name länger, werden überzählige Zeichen abgeschnitten.

Rückgabewert Der Rückgabewert gibt die tatsächliche Länge des in den Puffer kopierten Strings an.

Er ist 0, wenn das gewünschte Format nicht existiert oder zu den vordefinierten gehört.

GetClipboardOwner

Syntax
```
functionGetClipboardOwner: HWnd;
```

Beschreibung Diese Funktion ermittelt das Handle des Fensters, das aktueller Besitzer der Zwischenablage ist.

Die Funktion hat keine Parameter.

Rückgabewert Der Rückgabewert bezeichnet das Fenster, das die Zwischenablage besitzt.

Ist die Zwischenablage derzeit keinem Fenster zugeordnet, ist der Wert 0.

Hinweis Die Zwischenablage kann auch dann Daten enthalten, wenn sie momentan keinem Fenster zugeordnet ist.

GetClipboardViewer

Syntax
```
function GetClipboardViewer: HWnd;
```

Beschreibung Diese Funktion ermittelt das Handle des ersten Fensters in der Kette des Zwischenablage-Viewer.

Die Funktion hat keine Parameter.

Rückgabewert Der Rückgabewert bezeichnet das aktuell für die Darstellung der Zwischenablage verantwortliche Fenster. Existiert kein solches Fenster, ist er 0.

GetClipBox

Syntax
```
function GetClipBox(DC: HDC; var Rect: TRect): Integer;
```

Beschreibung Diese Funktion liefert die Größe des kleinstmöglichen, die aktuelle Clipping-Grenze umgebenden Rechtecks. Im Parameter *Rect* werden diese Daten gespeichert .

DC: Bezeichnet den Gerätekontext.

Rect: Eine *TRect*-Datenstruktur, in der die Größe des Rechtecks abgelegt wird.

Rückgabewert Der Rückgabewert gibt den Typ der Clipping-Region an. Folgende Werte sind möglich:

Wert	Bedeutung
ComplexRegion	Die Clipping-Region hat überlappende Grenzen.
Error	Der Gerätekontext ist ungültig.
NullRegion	Die Clipping-Region ist leer.
SimpleRegion	Die Clipping-Region hat keine überlappende Grenzen.

GetCodeHandle

Syntax
```
function GetCodeHandle(Proc: TFarProc): THandle;
```

Beschreibung Diese Funktion ermittelt, welches Code-Segment die Funktion enthält, auf die der Parameter *Proc* zeigt.

Proc: Die Adresse der Prozedurinstanz.

Rückgabewert Der Rückgabewert bezeichnet das Code-Segment, das die bezeichnete Funktion enthält.

Hinweis Ist das die Funktion enthaltende Code-Segment bereits geladen, wird es von der Funktion *GetCodeHandle* als »kürzlich benutzt« markiert, wenn nicht, versucht die Funktion, das Segment zu laden. Eine Anwendung kann daher diese Funktion verwenden, um ein oder mehrere Segment(e) für eine bestimmte Aufgabe im voraus zu laden.

GetCodeInfo

Syntax
```
procedure GetCodeInfo(Proc: TFarProc; SegInfo: Pointer);
```

Beschreibung Diese Prozedur ermittelt einen Zeiger auf ein Array von 16-Bit-Werten. Das Array enthält Informationen über das Code-Segment, das die Funktion enthält, auf die der Parameter *Proc* zeigt.

Proc: Die Adresse der Funktion im Segment, über das Informationen ermittelt werden sollen. Anstelle der *Segment:Offset*-Adresse kann der Wert auch die Form eines Modul-Handles und einer Segmentnummer haben. Die Funktion *GetModuleHandle* gibt das Handle eines benannten Moduls zurück.

SegInfo: Zeigt auf ein Feld mit vier 32-Bit-Werten, das mit den Informationen über das Code-Segment gefüllt wird.

Die Beschreibung der Werte in diesem Array finden Sie im folgenden Hinweis.

Hinweis Der Parameter *SegInfo* zeigt auf ein Array mit vier 32-Bit-Werten, das Informationen über Adresse und Größe des Segments und seine Attribute enthält. Die folgende Liste beschreibt die möglichen Werte:

Offset	Beschreibung
0	Bestimmt den Offset des virtuellen Sektors der Daten des Segments relativ zum Beginn der Datei in Bytes. Der Wert 0 bedeutet, daß keine Daten verfügbar sind.
2	Bestimmt die Größe des Segments in der Datei in Bytes. Der Wert 0 entspricht 64 KByte.
4	Enthält Flags, welche die Attribute des Segments angeben. Die folgende Liste beschreibt diese Flags: *Bit* *Bedeutung* 0-2 Bestimmt den Typ des Segments. Ist das Bit 0 auf 1 gesetzt, liegt ein Datensegment, sonst ein Code-Segment vor. 3 Bestimmt, ob die Daten des Segments wiederholt werden. Dies ist der Fall, wenn dieses Bit auf 1 gesetzt ist. 4 Bestimmt, ob ein verschiebbares oder festes Datensegment vorliegt. Das Segment ist verschiebbar, wenn das Bit auf 1 gesetzt ist. 5 Wird nicht zurückgegeben. 6 Wird nicht zurückgegeben. 7 Bestimmt, ob ein Read-Only-Datensegment oder ein Execute-Only-Code-Segment vorliegt. Ist das Bit auf 1 gesetzt und ist das Segment ein Code-Segment, handelt es sich um ein Execute-Only-Code-Segment. Ist das Bit dagegen auf 0 gesetzt und liegt ein Datensegment vor, handelt es sich um ein Read-Only-Datensegment. 8 Bestimmt, ob Relozierungsinformationen zu dem Segment existieren. Ist das Bit auf 1 gesetzt, sind entsprechende Informationen vorhanden. Sonst nicht. 9 Bestimmt, ob das Segment über Debug-Informationen verfügt. Ist das Bit auf 1 gesetzt, existieren solche Informationen. Sonst nicht. 10-11 Werden nicht zurückgegeben. 12-15 Werden nicht zurückgegeben.
6	Bestimmt die absolute Größe des für das Segment reservierten Speichers. Der Wert kann die tatsächliche Größe des Segments übersteigen. Der Wert 0 entspricht 65536 Bytes.

GetCommError

Syntax

```
function GetCommError(Cid: Integer; var Stat: TComStat): Integer;
```

Beschreibung Tritt ein Schnittstellenfehler auf, sperrt Windows die Schnittstelle, bis der Fehler durch die Funktion *GetCommError* beho-

ben wird. Die Funktion füllt den Statuspuffer, auf den der Parameter *Stat* zeigt, mit dem aktuellen Status der über den Parameter *Cid* angegeben Schnittstelle.

Außerdem wird der Fehlercode, der seit dem letzten Funktionsaufruf von *GetCommError* aufgetreten ist, zurückgeliefert. Hat *Stat* den Wert 0, wird nur der Fehlercode zurückgegeben.

Cid: Bestimmt, welche Schnittstelle überprüft werden soll. Die Funktion *OpenComm* gibt diesen Wert zurück.

Stat: Die *TComStat*-Struktur, die den Gerätestatus aufnehmen soll. Die Struktur enthält Informationen über eine Schnittstelle.

Rückgabewert

Der Rückgabewert bestimmt den Fehlercode der zuletzt ausgeführten Schnittstellenfunktion. Dabei kann es sich entweder um einen einzelnen Wert oder eine Kombination mehrerer Werte aus der folgenden Übersicht handeln.

Wert	Bedeutung
ce_Break	Die Hardware stellt eine Abbruchbedingung fest.
ce_CTSTO	Clear-to-send-Unterbrechung (time out). *CTS* ist für die durch *CtsTimeout* festgelegte Dauer gelöscht, während gleichzeitig versucht wird, ein Zeichen zu übertragen.
ce_DNS	Die parallele Schnittstelle ist nicht ausgewählt.
ce_DSRTO	Data-set-ready-Unterbrechung. *DSR* ist für die durch *DsrTimeout* festgelegte Dauer gelöscht, während gleichzeitig versucht wird, ein Zeichen zu übertragen.
ce_Frame	Die Hardware stellt einen Framing-Fehler fest.
ce_IOE	Ein I/O-Fehler tritt während des Versuchs der Kommunikation mit einer parallelen Schnittstelle auf.
ce_Mode	Der gewünschte Modus wird nicht unterstützt oder der Parameter *Cid* ist ungültig. Ist der Parameter gesetzt, ist dies der einzig mögliche Fehler.
ce_OOP	Der Drucker meldet Papierende.
ce_Overrun	Ein Zeichen wird von der Hardware nicht rechtzeitig gelesen. Das Zeichen ist verloren.
ce_PTO	Bei der Kommunikation mit der parallelen Schnittstelle tritt eine Unterbrechung auf.
ce_RLSDTO	Receive-line-signal-detect-Unterbrechung. *RLSD* ist für die durch *RlsdTimeout* festgelegte Dauer gelöscht, während gleichzeitig versucht wird, ein Zeichen zu übertragen.
ce_RxOver	Überlauf der Empfangs-Warteschlange. Entweder reicht der Platz in der Eingabe-Warteschlange nicht aus oder die Übertragung wurde nach Erhalt des *EofChar*-Zeichens fortgesetzt.

Wert	Bedeutung
ce_RxParity	Die Hardware stellt einen Parity-Fehler fest.
ce_TxFull	Die Übertragungs-Warteschlange ist voll. Gleichzeitig wird versucht, ein Zeichen zur Warteschlange hinzuzufügen.

Querverweis *OpenComm*

GetCommEventMask

Syntax
```
function GetCommEventMask(Cid, EvtMask: Integer): Word;
```

Beschreibung Diese Funktion ermittelt den Wert der aktuellen Ereignismaske und löscht sie anschließend. *GetCommEventMask* muß verwendet werden, um den Verlust eines Ereignisses zu verhindern.

Cid: Legt die zu untersuchende Schnittstelle fest. Die Funktion *OpenComm* gibt diesen Wert zurück.

EvtMask: Legt fest, welche Ereignisse unterstützt werden. Eine Liste der Ereigniswerte finden Sie in Kapitel 1 »Windows-Stile und -Konstanten« im Abschnitt »Schnittstellenereigniswerte«.

Rückgabewert Der Rückgabewert gibt den aktuellen Wert der Ereignismaske an. Jedes Bit der Ereignismaske zeigt an, ob ein vorgegebenes Ereignis vorgekommen ist. Ist dies der Fall, ist das Bit auf 1 gesetzt.

Querverweis *OpenComm, SetCommEventMask*

GetCommState

Syntax
```
function GetCommState(Cid: Integer; var DCB: TDCB):
Integer;
```

Beschreibung Diese Funktion füllt den Puffer mit dem Gerätesteuerblock, auf den der Parameter *DCB* zeigt. Die betroffene Schnittstelle wird durch den Parameter *Cid* bezeichnet.

Cid: Bestimmt die zu überprüfende Schnittstelle. Die Funktion *OpenComm* gibt diesen Wert zurück.

DCB: Eine *TDCB*-Datenstruktur, die den aktuellen Gerätesteuerblock aufnehmen soll. Die Struktur definiert die Steuerungseinstellungen des Geräts.

Rückgabewert Nach erfolgreicher Ausführung der Funktion ist der Rückgabewert 0, im Fall eines Fehlers ist der Rückgabewert negativ.

Querverweis *OpenComm*

GetCurrentPDB

Syntax
```
function GetCurrentPDB: Word;
```

Beschreibung Diese Funktion gibt die Paragraphenadresse oder den Selektor der aktuellen DOS Programmdatenbasis (PDB) zurück. Diese wird auch Program Segment Prefix (PSP) genannt.

Die Funktion hat keine Parameter.

Rückgabewert Der Rückgabewert ist die Paragraphenadresse oder der Selektor der aktuellen PDB.

GetCurrentPosition

Syntax
```
function GetCurrentPosition(DC: HDC): LongInt;
```

Beschreibung Diese Funktion ermittelt die virtuellen Koordinaten der aktuellen Position.

DC: Bezeichnet einen Gerätekontext.

Rückgabewert Der Rückgabewert gibt die aktuelle Position an. Die y-Koordinate liegt im höherwertigen Word, die x-Koordinate im niederwertigen.

GetCurrentTask

Syntax
```
function GetCurrentTask: THandle;
```

Beschreibung Diese Funktion gibt das Handle der gegenwärtig ausgeführten Task zurück. Die Funktion hat keine Parameter.

Rückgabewert Wenn die Funktion erfolgreich ausgeführt wurde, bezeichnet der Rückgabewert die Task, ansonsten ist er 0.

GetCurrentTime

Syntax
```
function GetCurrentTime: LongInt;
```

Beschreibung Diese Funktion ermittelt die aktuelle Windows-Zeit. Diese ist die Anzahl von Millisekunden, die seit dem Systemstart vergangen sind.

Die Funktion hat keine Parameter.

Rückgabewert Der Rückgabewert gibt die aktuelle Zeit in Millisekunden an.

Hinweis Die Funktionen *GetCurrentTime* und *GetMessageTime* geben unterschiedliche Zeitangaben zurück. *GetMessageTime* gibt die Windows-Zeit bei Erzeugung der angegebenen Botschaft zurück, nicht die aktuelle Windows-Zeit.

Ensteht in der Systemuhr ein Überlauf, wird sie auf 0 gesetzt.

GetCursorPos

Syntax
```
procedure GetCursorPos(var Point: TPoint);
```

Beschreibung Diese Prozedur ermittelt die aktuelle Mauszeigerposition in Form von Bildschirmkoordinaten. Dann werden diese in die *TPoint*-Struktur *Point* kopiert.

Point: Eine *TPoint*-Struktur, die die Bildschirmkoordinaten des Mauszeigers aufnehmen soll.

Hinweis Die Position des Mauszeigers wird immer in Form von Bildschirmkoordinaten angegeben. Sie wird vom Abbildungsmodus des Fensters, das den Mauszeiger enthält, nicht beeinflußt.

GetDC

Syntax
```
function GetDC(Wnd: HWnd): HDC;
```

Beschreibung Diese Funktion ermittelt ein Handle für einen Bildschirmkontext für den Client-Bereich eines gegebenen Fensters. Der Bildschirmkontext kann in nachfolgenden Funktionen der grafischen Geräteschnittstelle (GDI) zum Zeichnen im Client-Bereich verwendet werden.

GetDC stellt einen gemeinsamen, einen klassenspezifischen oder einen privaten Bildschirmkontext in Abhängigkeit vom Klassenstil des vorgegebenen Fensters zur Verfügung. Für gemeinsame Bildschirmkontexte weist *GetDC* dem Kontext bei jedem erneuten Aufruf Standardattribute zu. Bei klassenspezifischen und privaten werden die vorher zugewiesenen Attribute nicht verändert.

Wnd: Bezeichnet das Fenster, dessen Bildschirmkontext abgerufen werden soll.

Rückgabewert Wenn die Funktion erfolgreich ausgeführt wurde, bezeichnet der Rückgabewert den Bildschirmkontext des Client-Bereichs des gegebenen Fensters, andernfalls ist er 0.

Hinweis Nach dem Zeichnen mit einem gemeinsamen Bildschirmkontext muß die Funktion *ReleaseDC* verwendet werden, um den Kontext freizugeben.

Klassenspezifische und private Bildschirmkontexte müssen nicht freigegeben werden. Da zu einem bestimmten Zeitpunkt höchstens fünf Bildschirmkontexte verfügbar sind, ist die Freigabe der Kontexte nötig, um anderen Anwendungen den Zugriff auf einen Bildschirmkontext zu ermöglichen.

Querverweis *ReleaseDC*

GetDCOrg

Syntax
```
function GetDCOrg(DC: HDC): LongInt;
```

Beschreibung Diese Funktion ermittelt den Ursprung des Bildschirmkontextes. Dieser Ursprung bestimmt den von Windows verwendeten Offset zur Übersetzung von Gerätekoordinaten in Client-Koordinaten von Punkten in einem Anwendungsfenster. Der Ursprung ist relativ zum physikalischen Ursprung des Bildschirms gebildet.

DC: Bezeichnet den Gerätekontext, dessen Ursprung ermittelt werden soll.

Rückgabewert Der Rückgabewert gibt den Ursprung in Gerätekoordinaten an. Die y-Koordinate liegt im höherwertigen, die x-Koordinate im niederwertigen Word.

GetDesktopWindow

Syntax
```
function GetDesktopWindow: HWnd;
```

Beschreibung Diese Funktion liefert das Fenster-Handle für das Desktop-Fenster von Windows. Das Desktop-Fenster bedeckt den gesamten Bildschirm und ist die Oberfläche, auf der alle anderen Fenster und Symbole dargestellt werden.

Die Funktion hat keine Parameter.

Rückgabewert Der Rückgabewert bezeichnet das Desktop-Fenster von Windows.

GetDeviceCaps

Syntax
```
function GetDeviceCaps(DC: HDC; Index: Integer):
Integer;
```

Beschreibung Diese Funktion stellt gerätespezifische Informationen über einen gegebenen Bildschirm dar.

Der Parameter *Index* legt den Typ der gewünschten Informationen fest.

DC: Bezeichnet den Gerätekontext.

Index: Bestimmt die zurückzugebende Information. Es kann sich um einen beliebigen Wert der im Hinweis aufgeführten Tabelle handeln.

Rückgabewert Der Rückgabewert gibt den Wert der gewünschten Information an.

Hinweis Die folgende Tabelle beschreibt die möglichen Werte des Parameters *Index*.

Index	Bedeutung	
DriverVersion	Versionsnummer, z.B. $100 für 1.0.	
Technology	Gerätetechnik. Folgende Werte sind möglich:	
	dt_Plotter	Vektorplotter
	dt_RasDisplay	Rasterbildschirm
	dt_RasPrinter	Rasterdrucker
	dt_RasCamera	Rasterkamera
	dt_CharStream	Zeichen-Stream
	dt_MetaFile	Metafile
	dt_DispFile	Bildschirmdatei
HorzSize	Breite der physikalischen Anzeige in Millimetern.	
VertSize	Höhe der physikalischen Anzeige in Millimetern.	
HorzRes	Breite der Anzeige in Pixel.	
VertRes	Höhe der Anzeige in Scan-Zeilen.	
LogPixelsX	Anzahl der Pixel pro virtuellem Zoll über die Breite der Anzeige.	
LogPixelsY	Anzahl der Pixel pro virtuellem Zoll über die Höhe der Anzeige.	
BitsPixel	Anzahl benachbarter Farb-Bits eines Pixels.	
Planes	Anzahl der Farbebenen.	
NumBrushes	Anzahl der gerätespezifischen Pinsel.	
NumMarkers	Anzahl der Markierungssymbole.	
NumPens	Anzahl der gerätespezifischen Stifte.	
NumFonts	Anzahl der gerätespezifischen Schriften.	
NumColors	Anzahl der Einträge in der Farbtabelle des Geräts.	
AspectX	Relative Breite des zum Zeichnen einer Linie verwendeten Pixels eines Geräts.	
AspectY	Relative Höhe des zum Zeichnen einer Linie verwendeten Pixels eines Geräts.	
AspectXY	Diagonale Abmessung des zum Zeichnen einer Linie verwendeten Pixels eines Geräts.	
PDeviceSize	Größe der internen PDEVICE-Datenstruktur.	
ClipCaps	Flag, das die Clipping-Fähigkeit des Geräts anzeigt. Es ist 1, wenn das Gerät entlang eines Rechtecks abschneiden kann und 0, wenn es dazu nicht in der Lage ist.	
SizePalette	Anzahl der Einträge in der Systempalette. Dieser Index ist nur dann gültig, wenn der Gerätetreiber das *rc_Palette*-Bit im *RasterCaps*-Index setzt. Er ist nur verfügbar, wenn die Versionsnummer des Treibers 3.0 oder höher ist.	
NumReserved	Anzahl der reservierten Einträge in der Systempalette. Dieser Index ist nur dann gültig, wenn der Gerätetreiber das *rc_Palette*-Bit im *RasterCaps*-Index setzt. Er ist nur verfügbar, wenn die Versionsnummer des Treibers 3.0 oder höher ist.	

Index	Bedeutung	
ColorRes	Gegenwärtige Farbauflösung des Geräts in Bits je Pixel. Dieser Index ist nur dann gültig, wenn der Gerätetreiber das *rc_Palette*-Bit im *RasterCaps*-Index setzt. Er ist nur verfügbar, wenn die Versionsnummer des Treibers 3.0 oder höher ist.	
RasterCaps	Wert, der die Rasterfähigkeiten des Geräts anzeigt. Die Bedeutung der Werte zeigt folgende Liste:	
	rc_Banding	Benötigt Frequenzunterstützung.
	rc_BitBlt	Fähig zur Bitmap-Übertragung.
	rc_BigFont	Unterstützt Schriften mit mehr als 64 KByte.
	rc_Bitmap64	Fähig zur Unterstützung von Bitmaps mit mehr als 64 KByte.
	rc_DI_Bitmap	Fähig zur Unterstützung von *SetDIBits* und *GetDIBits*.
	rc_DIBToDev	Fähig zur Unterstützung von *SetDIBitsToDevice*.
	rc_FLOODFILL	Fähig zum Ausmalen.
	rc_GDI20_Output	Fähig zur Unterstützung von Merkmalen der Version Windows 2.0.
	rc_Palette	Gerät unterstützt Farbpaletten.
	rc_Scaling	Fähigkeit zum Skalieren.
	rc_StretchBlt	Fähig zur Durchführung von *StretchBlt*.
	rc_StretchDIB	Fähig zur Durchführung von *StretchDIBits*.
CurveCaps	Zeigt die Bogenzeichnungsfähigkeit des Geräts. Die Konstanten haben folgende Bedeutung:	
	cc_Chord	Gerät kann Kreisbögen darstellen.
	cc_Circles	Gerät kann Kreise darstellen.
	cc_Ellipses	Gerät kann Ellipsen darstellen.
	cc_Interiors	Gerät kann Ausfüllungen darstellen.
	cc_None	Gerät hat keine Bogenzeichnungsfähigkeiten.
	cc_Pie	Gerät kann Tortendiagramme darstellen.
	cc_Styled	Gerät kann verschiedene Linien darstellen.
	cc_Wide	Gerät kann breite Linien darstellen.
	cc_WideStyled	Gerät kann verschiedene breite Linien darstellen.
LineCaps	Zeigt die Linienzeichnungsfähigkeiten des Geräts. Die Konstanten haben folgende Bedeutung:	
	lc_Interiors	Gerät kann Ausfüllungen darstellen.

Index	Bedeutung	
	lc_Marker	Gerät kann Markierungen darstellen.
	lc_None	Gerät hat keine Linienzeichnungsfähigkeiten.
	lc_PolyLine	Gerät kann Linienzüge darstellen.
	lc_PolyMarker	Gerät kann Punktmarkierungen darstellen.
	lc_Styled	Gerät kann verschiedene Linien darstellen.
	lc_Wide	Gerät kann breite Linien darstellen.
	lc_WideStyled	Gerät kann verschiedene breite Linien darstellen.
PolygonalCaps	Zeigt die Fähigkeit des Geräts zum Zeichnen von Vielecken. Die Konstanten haben folgende Bedeutung:	
	pc_Interiors	Gerät kann Ausfüllungen darstellen.
	pc_None	Vielecke können nicht dargestellt werden.
	pc_Polygon	Gerät kann Vielecke darstellen.
	pc_Rectangle	Gerät kann Rechtecke darstellen.
	pc_ScanLine	Gerät kann Scan-Zeilen darstellen.
	pc_Styled	Gerät kann verschiedene Rahmen darstellen.
	pc_Trapezoid	Gerät kann Trapeze darstellen.
	pc_Wide	Gerät kann breite Rahmen darstellen.
	pc_Wide Styled	Gerät kann verschiedene breite Rahmen darstellen.
	pc_WindPolygon	Gerät kann komplexe Vielecke darstellen.
TextCaps	Zeigt die Textdarstellungsfähigkeiten des Geräts. Die Konstanten haben folgende Bedeutung:	
	tc_cp_Stroke	Gerät kann vektoriell mit Clipping ausgeben.
	tc_cr_90	Gerät kann Buchstaben um 90 Grad rotieren.
	tc_cr_Any	Gerät kann beliebige Buchstabenrotationen vornehmen.
	tc_ca_Double	Gerät kann Zeichen fett setzen.
	tc_la_Able	Gerät kann kursive Zeichen darstellen.

Index	Bedeutung	
	tc_op_Character	Gerät kann zeichengenau ausgeben.
	tc_op_Stroke	Gerät kann vektoriell ausgeben.
	tc_va_Able	Gerät kann Rasterschriften darstellen.
	tc_sa_Double	Gerät kann mit zweifachem Zeichenabstand skalieren.
	tc_sa_Integer	Gerät kann das Mehrfache eines Integerwerts zur Skalierung verwenden.
	tc_sa_Contin	Gerät kann einen beliebigen vielfachen Wert zur Skalierung verwenden.
	tc_st_X_YIndep	Gerät kann unabhängig von X und Y skalieren.
	tc_so_Able	Gerät kann Zeichen durchgestrichen darstellen.
	tc_ua_Able	Gerät kann unterstreichen.
	tc_va_Able	Gerät kann Vektorschriften darstellen.

GetDialogBaseUnits

Syntax
```
function GetDialogBaseUnits: LongInt;
```

Beschreibung Diese Funktion gibt die von Windows zur Erstellung von Dialogfenstern verwendeten Dialogbasiseinheiten zurück, die von der Anwendung zur Berechnung der durchschnittlichen Zeichenbreite einer Schrift verwendet werden sollten.

Die Funktion hat keine Parameter.

Rückgabewert Der Rückgabewert gibt die Dialogbasiseinheiten an.

Das höherwertige Word enthält die Höhe in Pixel, abgeleitet aus der Höhe der Systemschrift. Das niederwertige Word enthält die Breite in Pixel, abgeleitet aus der Breite der Systemschrift.

Hinweis Die zurückgegebenen Werte repräsentieren Dialogbasiseinheiten vor der Skalierung in aktuelle Dialogeinheiten.

Um die tatsächliche Höhe (H) und Breite (B) eines Steuerelements in Pixel festzustellen, muß folgende Formel mit vorgegebener Höhe (*x*) und Breite (*y*) in Dialogeinheiten sowie der Rückgabewert (Dialogbasiseinheit) der Funktion *GetDialogBaseUnits* verwendet werden:

B: = (*x* * *LoWord*(*Dialogbasiseinheit*))/4
H: = (*y* * *HiWord*(*Dialogbasiseinheit*))/8

Um Rundungsfehler zu vermeiden, sollte die Multiplikation vor der Division durchgeführt werden, außer wenn die Dialogbasiseinheiten ohne Rest durch vier teilbar sind.

GetDIBits

Syntax

```
function GetDIBits(DC: HDC; Bitmap: HBitmap; StartScan,
NumScans: Word; Bits: Pointer; var BitInfo:
TBitmapInfo; Usage: Word): Integer;
```

Beschreibung

Diese Funktion ermittelt die Bits des angegebenen Bitmaps und kopiert sie im geräteunabhängigen Format in den Puffer, auf den der Parameter *Bits* zeigt. *BitInfo* bezeichnet die Farbinformationen der geräteunabhängigen Bits in der *TBitmapInfo*-Datenstruktur.

DC: Legt den Gerätekontext fest.

Bitmap: Legt das Bitmap fest.

StartScan: Bestimmt die Anordnung der ersten Scan-Zeile in dem durch *Bits* bezeichneten Bitmap.

NumScans: Bestimmt die Anzahl der zu kopierenden Zeilen.

Bits: Zeigt auf den Puffer, der die Bitmap-Daten im geräteunabhängigen Format aufnimmt.

BitInfo: Eine *TBitmapInfo*-Datenstruktur, die das Farbformat und die Dimensionierung des geräteunabhängigen Bitmaps enthält.

Usage: Legt fest, ob die Felder *bmiColors* des Parameters *BitInfo* Indexeinträge der aktuellen virtuellen Farbpalette oder explizite RGB-Werte enthalten. Der Parameter *Usage* muß einen der folgenden Werte annehmen:

Wert	Bedeutung
DIB_Pal_Colors	Die Farbtabelle muß aus einem Array von 16-Bit-Indizen bestehen, die in die aktuell realisierte virtuelle Farbpalette verweisen.
DIB_RGB_Colors	Die Farbtabelle enthält explizite RGB-Werte.

Rückgabewert

Der Rückgabewert bestimmt die Anzahl der Scan-Zeilen, die aus dem Bitmap kopiert wurden. Er ist 0, wenn ein Fehler aufgetreten ist.

Hinweis

Hat der Parameter *Bits* den Wert **nil**, füllt *GetDIBits* die *TBitmapInfo*-Datenstruktur aus, auf die *BitInfo* zeigt, die Bits aus dem Bitmap werden aber nicht gespeichert.

Das durch den Parameter *Bitmap* bezeichnete Bitmap darf

durch die Anwendung während dieses Funktionsaufrufs in keinen Gerätekontext ausgewählt sein.

Ursprung des geräteunabhängigen Bitmaps ist dessen untere linke Ecke, die obere linke Ecke ist der Ursprung, wenn der Abbildungsmodus auf *mm_Text* gesetzt ist.

GetDlgCtrlID

Syntax
```
function GetDlgCtrlID(Wnd: HWnd): Integer;
```

Beschreibung Diese Funktion gibt den ID-Wert des durch den Parameter *Wnd* bezeichneten, untergeordneten Fensters zurück.

Wnd: Bezeichnet das untergeordnete Fenster.

Rückgabewert Ist die Funktion erfolgreich, entspricht der Rückgabewert dem numerischen Bezeichner des untergeordneten Fensters. Wenn sie fehlschlägt oder der Parameter *Wnd* kein gültiges Fenster-Handle darstellt, ist der Rückgabewert 0.

Hinweis Da die Fenster der Hauptebene keinen ID-Wert haben, ist der Rückgabewert ungültig, wenn der Parameter *Wnd* auf ein solches Fenster verweist.

GetDlgItem

Syntax
```
function GetDlgItem(Dlg: HWnd; IDDlgItem: Integer):
HWnd;
```

Beschreibung Diese Funktion liefert das Handle eines im Dialogfenster enthaltenen Steuerelements. Das Dialogfenster wird durch den Parameter *Dlg* bestimmt.

Dlg: Bezeichnet das Dialogfenster, in dem das Steuerelement enthalten ist.

IDDlgItem: Bestimmt die Integer-ID des zu ermittelnden Elements.

Rückgabewert Der Rückgabewert bezeichnet das vorgegebene Steuerelement. Er ist 0, wenn kein Steuerelement mit der durch *IDDlgItem* gegebenen Integer-ID existiert.

Hinweis Die Funktion *GetDlgItem* kann für jedes Paar über- und untergeordneter Fenster verwendet werden, sie ist also nicht nur bei Dialogfenstern anwendbar. Wenn *Dlg* ein übergeordnetes Fenster bezeichnet und das untergeordnete Fenster eine eindeutige ID besitzt, liefert *GetDlgItem* ein gültiges Handle für das untergeordnete Fenster.

Die ID des untergeordneten Fensters wurde durch den Parameter *Menu* der erzeugenden Funktion *CreateWindow* festgelegt.

GetDlgItemInt

Syntax
```
function GetDlgItemInt(Dlg: HWnd; IDDlgItem: Integer;
Translate: PBool; Signed: Bool): Word;
```

Beschreibung Diese Funktion übersetzt den Text eines Steuerelements des gegebenen Dialogfensters in einen Integer.

GetDlgItemInt ermittelt den Text des durch den Parameter *IDDlgItem* bezeichneten Steuerelements und übersetzt diesen ohne Berücksichtigung etwaiger Leerzeichen am Anfang des Textes durch Konvertierung der Dezimalziffern.

Die Übersetzung wird beendet, wenn das Ende des Textes erreicht ist oder ein nichtnumerisches Zeichen aufgefunden wird.

Ist der Parameter *Signed True*, prüft *GetDlgItemInt* auf ein Minuszeichen (-) am Anfang des Textes. Ist dieses vorhanden, so wird der Text in eine vorzeichenbehaftete Zahl übersetzt, andernfalls ist die Zahl vorzeichenlos.

GetDlgItemInt gibt 0 zurück, wenn die übersetzte Zahl bei negativen Werten größer als 32767 und bei positiven Werten größer als 65535 ist. Treten Fehler auf, wie z.B. das Auffinden eines nichtnumerischen Zeichens oder das Überschreiten des vorgegebenen Höchstwerts, kopiert die Funktion *False* in das Ziel, auf das der Parameter *Translate* zeigt. Treten keine Fehler auf, erhält der Paramater *Translate* den Wert *True*. Ist *Translate False*, werden keine Warnungen ausgegeben.

Die Funktion übergibt eine *wm_GetText*-Botschaft an das Steuerelement.

Dlg: Bezeichnet das Dialogfenster.

IDDlgItem: Bestimmt den Integer-Bezeichner des zu übersetzenden Objekts des Dialogfensters.

Translate: Zurückgegebener Bool-Wert. Er ist *False*, wenn bei der Übersetzung ein Fehler auftritt.

Signed: Bestimmt, ob der zu ermittelnde Wert vorzeichenbehaftet ist.

Rückgabewert Der Rückgabewert gibt den übersetzten Wert des Textes des Dialogfensterelements an. Da 0 ein gültiger Rückgabewert ist, muß der Parameter *Translate* zum Auffinden von Fehlern verwendet werden.

Querverweis *wm_GetText*

GetDlgItemText

Syntax
```
function GetDlgItemText(Dlg: HWnd; IDDlgItem: Integer;
Str: PChar; MaxCount: Integer): Integer;
```

Beschreibung Diese Funktion ermittelt den zu einem Steuerelement in einem Dialogfenster gehörenden Text bzw. Überschrift und kopiert den Text an die durch den Parameter *Str* bezeichnete Position. Anschließend wird eine Zahl zurückgegeben, die der Anzahl der kopierten Zeichen entspricht.

GetDlgItemText übergibt eine *wm_GetText*-Botschaft an das Steuerelement.

Dlg: Bezeichnet das Dialogfenster mit dem gewünschten Steuerelement.

IDDlgItem: Bestimmt den Integer-Bezeichner des Objekts im Dialogfenster, dessen Überschrift oder Text ermittelt werden soll.

Str: Zeigt auf den Puffer, der den Text aufnimmt.

MaxCount: Bestimmt die maximale Länge des in *Str* zu kopierenden Strings in Bytes. Ist der String länger, als in *MaxCount* angegeben, werden die überzähligen Zeichen abgeschnitten.

Rückgabewert Der Rückgabewert gibt die Anzahl der tatsächlich in den Puffer kopierten Zeichen an. Er ist 0, wenn kein Text kopiert wurde.

Querverweis *wm_GetText*

GetDOSEnvironment

Syntax `function GetDOSEnvironment: PChar;`

Beschreibung Diese Funktion gibt einen Zeiger auf den Environment-String der laufenden Task zurück. Weitere Informationen über Format und Inhalt des Environment-Strings finden Sie in der *Microsoft MS-DOS Programmer's Reference.*

Die Funktion hat keine Parameter.

Hinweis Im Unterschied zu einer normalen Anwendung besitzt eine DLL (dynamische Linkbibliothek) keine Kopie des Environment-Strings. Aus diesem Grund muß sie *GetDOSEnvironment* zur Ermittlung des Strings aufrufen.

GetDoubleClickTime

Syntax `function GetDoubleClickTime: Word;`

Beschreibung Diese Funktion ermittelt die Zeit, die für einen Doppelklick der Maus zur Verfügung steht. Ein Doppelklick ist die kurz aufeinanderfolgende, zweimalige Betätigung der Maustaste, wobei die zweite Betätigung innerhalb einer festgelegten Zeit erfolgen muß. Die Doppelklickzeit ist die Anzahl von Millisekunden, die maximal zwischen zwei Tastenbetätigungen verstreichen darf. Die Funktion hat keine Parameter.

Rückgabewert Der Rückgabewert gibt die aktuelle Doppelklickzeit in Millisekunden an.

GetDriveType

Syntax
```
function GetDriveType(Drive: Integer): Word;
```

Beschreibung Diese Funktion stellt fest, ob in einem Laufwerk austauschbare oder fest installierte Speichermedien verwendet werden oder ob es sich um ein Netzlaufwerk handelt.

Drive: Bestimmt das Laufwerk, dessen Typ ermittelt werden soll. Laufwerk A: hat den Wert 0, B: den Wert 1, Laufwerk C: den Wert 2 usw.

Rückgabewert Der Rückgabewert gibt den Laufwerkstyp an. Möglich sind folgende Werte:

Wert	Bedeutung
Drive_Removeable	Diskettenlaufwerk.
Drive_Fixed	Festplattenlaufwerk.
Drive_Remote	Netzlaufwerk.

Der Rückgabewert ist 0, wenn die Funktion den Laufwerkstyp nicht ermitteln kann, und 1, wenn das angegebene Laufwerk nicht existiert.

GetEnvironment

Syntax
```
function GetEnvironment(PortName: PChar; Environ:
Pointer; MaxCount: Word): Integer;
```

Beschreibung Diese Funktion ermittelt die aktuelle Umgebung der an eine durch den Parameter *PortName* bezeichneten Systemschnittstelle angeschlossenen Geräteeinheit. Die Information wird in den Puffer kopiert, auf den der Parameter *Environ* zeigt.

Die von der grafischen Geräteschnittstelle (GDI) unterstützte Umgebung enthält binäre Daten, die von der GDI zur Erstellung eines Gerätekontexts für die Geräteeinheit an der gegebenen Schnittstelle verwendet werden.

Die Funktion führt nicht zum gewünschten Ergebnis, wenn keine Umgebung der vorgegebenen Schnittstelle aufgefunden werden kann.

Anwendungen können die Funktion mit dem auf 0 gesetzten Parameter *Environ* aufrufen, um die nötige Größe des Puffers für die Umgebung festzustellen. Anschließend kann für den Puffer Speicher reserviert und ein weiteres Mal die Funktion *GetEnvironment* aufgerufen werden, um die Umgebung zu ermitteln.

PortName: Zeigt auf den nullterminierten String, der den Namen der gewünschten Schnittstelle enthält.

Environ: Zeigt auf den Puffer, der die Umgebung aufnehmen soll.

MaxCount: Bestimmt die maximale Anzahl der in den Puffer zu kopierenden Bytes.

Rückgabewert Der Rückgabewert ist die Anzahl der in *Environ* kopierten Bytes. Ist der Parameter *Environ* auf 0 gesetzt, wird die zum Speichern der Umgebung benötigte Größe des Puffers zurückgegeben. Kann die Umgebung nicht gefunden werden, ist der Rückgabewert 0.

Hinweis Das erste Feld des Puffers, auf das der Parameter *Environ* zeigt, muß identisch mit dem im Parameter *DeviceName* an die Funktion *CreateDC* übergebenen Wert sein.

Bestimmt *PortName* eine Null-Schnittstelle (wie sie in der Datei WIN.INI definiert ist), wird der durch *Environ* bezeichnete Gerätename verwendet, um die gewünschte Umgebung aufzufinden.

GetFocus

Syntax
```
function GetFocus: HWnd;
```

Beschreibung Diese Funktion ermittelt das Handle des Fensters, das aktuell den Fokus besitzt.

Diese Funktion hat keine Parameter.

Rückgabewert Der Rückgabewert bezeichnet das Fenster, das akuell den Fokus besitzt. Konnte die Funktion nicht erfolgreich ausgeführt werden, ist der Rückgabewert 0.

GetFreeSpace

Syntax
```
function GetFreeSpace(Flag: Word): LongInt;
```

Beschreibung Diese Funktion untersucht den globalen Heap und gibt den zur Verfügung stehenden Speicher in Bytes zurück.

Flag: Bestimmt, ob der Heap über oder unter der EMS-Speicherabgrenzung in Large-Frame- und Small-Frame-EMS-Systemen geprüft wird. Ist der Parameter auf *gmem_Not_Banked* gesetzt, gibt *GetFreeSpace* die Größe des verfügbaren Speichers unterhalb der Abgrenzung zurück. Der Wert oberhalb der Abgrenzung wird zurückgegeben, wenn *Flag* den Wert 0 hat. In Systemen ohne EMS wird dieser Parameter ignoriert.

Rückgabewert Der Rückgabewert entspricht der Größe des verfügbaren Speichers in Byte. Dieser Speicher muß nicht notwendigerweise

zusammenhängend sein. Die Funktion *GlobalCompact* ermittelt die Anzahl der Bytes des größten zusammenhängenden Blocks im globalen Speicher.

Querverweis *GlobalCompact*

GetGValue

Syntax
```
function GetGValue(RGB: LongInt): Byte;
```

Beschreibung Diese Inline-Funktion ermittelt den grünen Farbanteil eines RGB-Farbwerts.

RGB: Gibt ein rotes, ein grünes und ein blaues Farbfeld an. Diese Felder bestimmen die Intensität der gegebenen Farbe.

Rückgabewert Der Rückgabewert ist ein Byte, das den grünen Farbwert des Parameters *RGB* enthält.

Hinweis Der Wert 255 entspricht der maximalen Intensität eines einzelnen Bytes, 0 entspricht der minimalen Intensität eines einzelnen Bytes.

GetInputState

Syntax
```
function GetInputState: Bool;
```

Beschreibung Diese Funktion untersucht, ob in der System-Warteschlange Maus-, Tastatur- oder Timer-Ereignisse zur Bearbeitung vorliegen.

Ein Ereignis ist ein Record, der Eingaben auf der Interrupt-Ebene beschreibt. Mausereignisse liegen bei Mausbewegungen oder Mausklicks vor. Ein Tastaturereignis ist jede Betätigung einer oder mehrer Tasten. Timer-Ereignisse liegen nach einer vorgegebenen Anzahl von Clock-Ticks vor. Solche Ereignisse werden von Windows in der System-Warteschlange gespeichert.

Die Funktion hat keine Parameter.

Rückgabewert Der Rückgabewert bestimmt, ob Maus-, Tastatur- oder Timer-Ereignisse vorliegen. Ist dies nicht der Fall, ist der Rückgabewert *False*.

GetInstanceData

Syntax
```
function GetInstanceData(Instance: THandle; Data: Word;
Count: Integer): Integer;
```

Beschreibung Diese Funktion kopiert Daten einer vorhergehenden Instanz der Anwendung in den Datenbereich der aktuellen Instanz. Der Parameter *Instance* legt fest, aus welcher Instanz die Daten

in den durch *Data* bezeichneten Bereich kopiert werden sollen. Der Parameter *Count* bestimmt die Anzahl der zu kopierenden Bytes.

Instance: Bezeichnet eine vorhergehende Instanz der Anwendung.

Data: Zeigt auf einen Puffer in der aktuellen Instanz.

Count: Bestimmt die Anzahl der zu kopierenden Bytes.

Rückgabewert Der Rückgabewert entspricht der Anzahl der tatsächlich kopierten Bytes.

GetKBCodePage

Syntax
```
function GetKBCodePage: Integer;
```

Beschreibung Diese Funktion ermittelt, welche OEM/ANSI-Tabelle von Windows geladen ist. Diese Funktion hat keine Parameter.

Rückgabewert Der Rückgabewert gibt die momentan von Windows geladene Code-Seite an.
Möglich sind die in folgender Tabelle aufgeführten Werte:

Wert	Bedeutung
437	Standard (USA, wird in den meisten Ländern verwendet. Zeigt an, daß sich keine Datei OEMANSI.BIN im Windows-Verzeichnis befindet).
850	International (OEMANSI.BIN = XLAT850.BIN)
860	Portugal (OEMANSI.BIN = XLAT860.BIN)
861	Island (OEMANSI.BIN = XLAT861.BIN)
863	Franko-Kanadisch (OEMANSI.BIN = XLAT863.BIN)
865	Norwegen/Dänemark (OEMANSI.BIN = XLAT865.BIN)

Hinweis Befindet sich die Datei OEMANSI.BIN im Windows-Verzeichnis, wird sie von Windows gelesen. Die OEM/ANSI-Übersetzungstabellen im Tastaturtreiber werden von ihr überschrieben.

Wählt der Anwender innerhalb des Setup-Programms eine Sprache, die nicht die Standard-Code-Seite (437) benutzt, kopiert Setup die passende Datei (zum Beispiel XLATPO.BIN) in OEMANSI.BIN im Windows-Systemverzeichnis.

Wird die Standard-Code-Seite verwendet, löscht Setup die Datei OEMANSI.BIN aus dem Windows-Systemverzeichnis.

GetKeyboardState

Syntax
```
procedure GetKeyboardState(var KeyState:
TKeyboardState);
```

Beschreibung Diese Prozedur kopiert den Status der 256 möglichen Tasten der virtuellen Tastatur in den durch *KeyState* bezeichneten Puffer. Das obere Bit eines jeden Bytes wird auf 1 gesetzt, wenn die Taste gedrückt ist, auf 0, wenn die Taste nicht betätigt wird. Das untere Bit wird auf 1 gesetzt, wenn die Taste seit dem Startup eine ungerade Zahl mal betätigt wurde, andernfalls enthält es den Wert 0.

KeyState: Ein 256-Byte-Puffer für die virtuellen Tasten-Codes.

Hinweis Eine Anwendung ruft die Prozedur *GetKeyboardState* als Reaktion auf eine Tatstatureingabebotschaft auf. Diese Prozedur ermittelt den Zustand der Tastatur bei Erstellung der Botschaft.

Um Informationen über einzelne Tasten zu erhalten, müssen folgende Schritte durchgeführt werden:

1. Erzeugen eines Zeichen-Arrays mit 256 Byte Länge.

2. Kopieren des durch *KeyState* bezeichneten Pufferinhalts in dieses Array.

3. Überprüfung der virtuellen Tasten-Codes, um den Status einzelner Tasten zu ermitteln.

GetKeyboardType

Syntax
```
function GetKeyboardType(TypeFlag: Integer): Integer;
```

Beschreibung Diese Funktion ermittelt den Typ der vom System verwendeten Tastatur.

TypeFlag: Bestimmt, ob der Rückgabewert der Funktion den Typ oder den Untertyp der Tastatur anzeigt. Möglich sind die in der folgenden Tabelle aufgeführten Werte:

Wert	Bedeutung
0	Die Funktion gibt den Typ der Tastatur zurück.
1	Die Funktion gibt den Untertyp der Tastatur zurück.
2	Die Funktion gibt die Anzahl der Funktionstasten der Tastatur zurück.

Rückgabewert Der Rückgabewert zeigt den Typ oder Untertyp der vom System verwendeten Tastatur oder die Anzahl der existierenden Funktionstasten an. Der Untertyp ist ein OEM-abhängiger Wert. Folgende Werte sind möglich:

Wert	Bedeutung
1	IBM PC/XT oder kompatible Tastatur (83 Tasten)
2	Olivetti M24 "ICO" Tastatur (102 Tasten)
3	IBM AT oder vergleichbare Tastatur (84 Tasten)

Wert	Bedeutung
4	IBM Erweiterte Tastatur (101 oder 102 Tasten)
5	Nokia 1050 und vergleichbare Tastaturen
6	Nokia 9140 und vergleichbare Tastaturen

Der Rückgabewert ist 0, wenn die Funktion fehlschlägt, oder der Parameter *TypeFlag* einen Wert größer 2 hat.

Hinweis

Eine Anwendung kann die Anzahl der Funktionstasten auf der Tastatur über deren Typ ermitteln.

Die folgende Tabelle zeigt die Anzahl der Funktionstasten jedes Tastaturtyps:

Typ	Anzahl der Funktionstasten
1	10
2	12 (ausnahmsweise 18)
3	10
4	12
5	10
6	24

GetKeyNameText

Syntax

```
function GetKeyNameText(lParam: LongInt; Buffer: PChar;
Size: Integer): Integer;
```

Beschreibung

Diese Funktion ermittelt einen String, der den Namen einer Taste enthält.

Der Tastaturtreiber enthält eine Liste von Namen in Form von Zeichenketten für die Tasten, deren Namen länger als ein einzelnes Zeichen sind. Der Tastenname wird entsprechend der Tastenanordnung der installierten Tastatur übersetzt. Die Übersetzung wird entsprechend der durch den Tastaturtreiber unterstützten Sprache vorgenommen.

lParam: Bestimmt den 32-Bit-Parameter der Tastaturbotschaft, die die Funktion gerade bearbeitet (z.B. *wm_KeyDown*). Byte 3 (Bits 16 bis 23) dieses Parameters ist ein Scan-Code. Bit 20 ist ein zusätzliches Bit, das einige Tasten auf einer erweiterten Tastatur unterscheidet. Bit 21 wird von der Anwendung gesetzt, um die Funktion anzuweisen, nicht zwischen linken oder rechten Ctrl - bzw. Shift -Tasten (etc.) zu unterscheiden.

Buffer: Bestimmt den nullterminierten String, der den Tastennamen aufnehmen soll.

Size: Bestimmt die maximale Länge des Tastennamens in Byte. In diesen Wert ist die abschließende 0 nicht eingeschlossen.

Rückgabewert Der Rückgabewert entspricht der tatsächlichen Länge des in *Buffer* kopierten Strings.

GetKeyState

Syntax
```
function GetKeyState(VirtKey: Integer): Integer;
```

Beschreibung Diese Funktion ermittelt den Status der durch den Parameter *VirtKey* bestimmten virtuellen Taste. Der Status zeigt an, ob die Taste gedrückt oder arretiert ist.

VirtKey: Bestimmt eine virtuelle Taste. Handelt es sich bei der angeforderten Taste um einen Buchstaben oder um eine Ziffer (A bis Z, a bis z, 0 bis 9), muß *VirtKey* der ASCII-Wert des Zeichens zugewiesen werden. Für die anderen Tasten muß ein gültiger virtueller Tasten-Code verwendet werden.

Rückgabewert Der Rückgabewert gibt den Status der vorgegebenen virtuellen Taste an. Ist das höherwertige Bit 1, wird die Taste gedrückt. Ist das niederwertige Bit 1, ist die Taste arretiert. Eine Arretierungstaste wie CapsLock ist arretiert, wenn sie seit dem Systemstart eine ungerade Anzahl mal betätigt wurde. Ist das Bit 0, ist die Taste nicht arretiert.

Hinweis Eine Anwendung ruft *GetKeyState* als Reaktion auf eine Tastatureingabebotschaft auf.

Die Funktion ermittelt den Status der Taste, den sie während der Erzeugung der Botschaft hatte.

GetLastActivePopup

Syntax
```
function GetLastActivePopup(Owner: HWnd): HWnd;
```

Beschreibung Diese Funktion ermittelt, welches zu dem durch *Owner* bezeichneten Fenster gehörige Pop-up-Fenster zuletzt aktiviert war.

Owner: Bezeichnet das besitzende Fenster.

Rückgabewert Der Rückgabewert bezeichnet das zuletzt aktivierte Pop-up-Fenster.

Der Rückgabewert ist identisch mit dem Parameter *Owner*, wenn eine der folgenden Bedingungen zutrifft:

- Das durch *Owner* bezeichnete Fenster war selbst das zuletzt aktivierte.
- Das durch *Owner* bezeichnete Fenster hat keine Pop-up-Fenster.
- Das durch *Owner* bezeichnete Fenster ist kein Hauptfenster oder es gehört zu einem anderen Fenster.

GetMapMode

Syntax `function GetMapMode(DC: HDC): Integer;`

Beschreibung Diese Funktion ermittelt den aktuellen Abbildungsmodus. Die Beschreibung der möglichen Werte finden Sie in Kapitel 1 »Windows-Stile und -Konstanten« im Abschnitt »mm_Xxxx«.

DC: Bezeichnet den Gerätekontext.

Rückgabewert Der Rückgabewert bestimmt den Abbildungsmodus.

Querverweis *SetMapMode*

GetMenu

Syntax `function GetMenu(Wnd: HWnd):HMenu;`

Beschreibung Ermittelt das Handle eines Menüs des gegebenen Fensters.

Wnd: Bezeichnet das Fenster, dessen Menü untersucht werden soll.

Rückgabewert Der Rückgabewert bezeichnet das Menü. Er ist 0, wenn das Fenster kein Menü besitzt. Wenn das Fenster untergeordnet ist, ist der Rückgabewert nicht definiert.

GetMenuCheckMarkDimensions

Syntax `function GetMenuCheckMarkDimensions: LongInt;`

Beschreibung Diese Funktion gibt die Größe des Bitmaps der Standardauswahlmarkierung (Häkchen) zurück. Windows stellt dieses Bitmap neben den markierten Menüeinträgen dar. Vor der Ersetzung des Häkchens durch die Funktion *SetMenuItemBitmaps* sollte die Anwendung die korrekte Größe der Bitmaps mit *GetMenuCheckMarkDimensions* ermitteln.

Die Funktion hat keine Parameter.

Rückgabewert Der Rückgabewert bestimmt Höhe und Breite des Bitmaps der Standardauswahlmarkierung. Das höherwertige Word enthält die Höhe, das niederwertige die Breite in Pixel.

Querverweis *SetMenuItemBitmaps*

GetMenuItemCount

Syntax `function GetMenuItemCount(Menu: Hmenu): Word;`

Beschreibung Diese Funktion stellt die Anzahl der Einträge in dem durch *Menu* bezeichneten Menü fest. Es kann sich um ein Pop-up-Menü oder ein Hauptmenü handeln.

Menu: Bezeichnet das Handle des zu untersuchenden Menüs.

Rückgabewert Der Rückgabewert gibt die Anzahl der Menüeinträge im durch *Menu* bezeichneten Menü an. Wenn die Funktion nicht erfolgreich ausgeführt werden konnte, ist der Rückgabewert -1.

GetMenuItemID

Syntax
```
function GetMenuItemID(Menu: HMenu; Pos: Integer): Word;
```

Beschreibung Diese Funktion ermittelt den Bezeichner des Menüeintrags, der sich an der durch *Pos* bezeichneten Position befindet.

Menu: Bezeichnet das Handle des Pop-up-Menüs, das den Eintrag enthält, dessen Bezeichner ermittelt wird.

Pos: Gibt die Position des zum Bezeichner gehörenden Menüeintrags (ausgehend von 0) an.

Rückgabewert Wenn die Funktion erfolgreich ausgeführt wurde, gibt der Rückgabewert die ID eines Eintrags in einem Pop-up-Menü an. Hat der Parameter *Menu* den Wert 0 oder ist der betreffende Eintrag selbst ein Pop-up-Menü, ist der Rückgabewert -1.

GetMenuState

Syntax
```
function GetMenuState(Menu: HMenu; Id, Flags: Word):
Word;
```

Beschreibung Diese Funktion ermittelt die Anzahl der Einträge im Pop-up-Menü, das mit dem durch *Id* bezeichneten Menüeintrag verbunden ist.

Voraussetzung ist, daß der Parameter *Menu* ein zu einem Pop-up-Menü gehöriges Menü angibt. Bezeichnet *Menu* ein Pop-up-Menü, ermittelt die Funktion den Status des durch *Id* bezeichneten Menüeintrags.

Menu: Bezeichnet das Menü.

Id: Bestimmt die *Id* des Menüeintrags.

Flags: Bestimmt die Interpretation des Parameters *Id*. Enthält der Parameter *Flags mf_ByPosition*, legt *wId* eine von 0 ausgehende, relative Position fest.

Enthält *Flags mf_ByCommand*, bestimmt *Id* die ID des Eintrags.

Rückgabewert Der Rückgabewert ist -1, wenn der angegebene Eintrag nicht existiert. Existiert auch das Menü nicht, wird ein Exit (mit schwerem Fehler) ausgelöst.

Bezeichnet *Id* ein Pop-up-Menü, enthält der Rückgabewert die Anzahl der Menüeinträge im höherwertigen Byte und die

zum Menü gehörenden Flags im niederwertigen Byte. Andernfalls liegt eine Maske mit dem logischen **or** verknüpfter Werte aus folgender Liste vor. Diese Maske beschreibt den Status des durch *Id* bestimmten Menüeintrags:

Wert	Bedeutung
mf_Checked	Auswahlmarkierung neben dem Eintrag (nur bei Pop-up-Menüs).
mf_Disabled	Eintrag ist deaktiviert.
mf_Enabled	Eintrag kann angewählt werden.
mf_Grayed	Eintrag ist in einen nicht aktivierbaren Zustand versetzt und grau hinterlegt.
mf_MenuBarBreak	Wie *mf_MenuBreak*, nur wird bei Pop-up-Menüs die neue Spalte durch eine vertikale Linie von der alten getrennt.
mf_MenuBreak	Eintrag ist in einer neuen Zeile (statisches Menü) oder einer neuen Spalte (Pop-up-Menü) ohne Trennung der Spalten positioniert.
mf_Separator	Horizontale Trennlinie wird gezeichnet (nur bei Pop-up-Menüs). Diese Linie kann nicht angewählt, markiert, in nicht aktivierbaren Zustand versetzt oder hervorgehoben werden.
mf_Unchecked	Keine Auswahlmarkierung neben dem Eintrag (Standardeinstellung).

GetMenuString

Syntax

```
function GetMenuString(Menu: HMenu; IDItem: Word; Str:
PChar; MaxCount: Integer; Flag: Word): Integer;
```

Beschreibung

Diese Funktion kopiert das Label des angegebenen Menüeintrags in den Parameter *Str*.

Menu: Bezeichnet das Menü.

IDItem: Bestimmt den Bezeichner (Integer) des Menüeintrags aus der Ressourcendatei oder den Offset des Menüeintrags im Menü in Abhängigkeit vom Wert des Parameters *Flag*.

Str: Zeigt auf den nullterminierten String, der das Label aufnimmt.

MaxCount: Bestimmt die maximale Länge des zu kopierenden Labels. Ist das Label länger, werden die überzähligen Zeichen abgeschnitten.

Flag: Bestimmt den Typ des Parameters *IDItem*. Enthält *Flags mf_ByPosition*, gibt *IDItem* eine von 0 ausgehende, relative Position an, enthält *wFlags mf_ByCommand*, gibt der Parameter *IDItem* die ID des Eintrags an.

Rückgabewert Der Rückgabewert entspricht der Anzahl der tatsächlich in den Puffer kopierten Bytes.

Hinweis Der Parameter *MaxCount* sollte die Anzahl der Zeichen im Label um 1 überschreiten, damit die den String abschließende 0 aufgenommen werden kann.

GetMessage

Syntax
```
function GetMessage(var Msg: TMsg; Wnd: HWnd;
MsgFilterMin, MsgFilterMax: Word): Bool;
```

Beschreibung Diese Funktion ruft eine Botschaft von der Anwendungs-Warteschlange ab und legt diese Botschaft in der *TMsg*-Struktur *Msg* ab.

Ist keine Botschaft verfügbar, übergibt *GetMessage* die Steuerung an andere Anwendungen, bis eine Botschaft verfügbar wird.

GetMessage ruft nur Botschaften ab, die dem durch den Parameter *Wnd* bezeichneten Fenster zugeordnet sind und die innerhalb des durch die Parameter *MsgFilterMin* und *MsgFilterMax* vorgeben Bereichs von Botschaftswerten liegen. Wenn *Wnd* 0 ist, ruft *GetMessage* Botschaften für jedes Fenster ab, das zur aufrufenden Anwendung gehört. (Die Funktion *GetMessage* ruft keine Botschaften für Fenster ab, die zu einer anderen Anwendung gehören). Sind *MsgFilterMin* und *MsgFilterMax* beide 0, gibt *GetMessage* alle verfügbaren Botschaften zurück (es wird keine Filterung durchgeführt).

Die Konstanten *wm_KeyFirst* und *wm_KeyLast* können als Filterwerte benutzt werden, um alle Botschaften abzurufen, die sich auf Tastatureingaben beziehen.

Die Konstanten *wm_MouseFirst* und *wm_MouseLast* können benutzt werden, um alle mausbezogenen Botschaften abzurufen.

Msg: Eine *TMsg*-Datenstruktur, die Botschaftsinformationen aus der Windows-Anwendungs-Warteschlange enthält.

Wnd: Bezeichnet das Fenster, dessen Botschaften überprüft werden sollen. Wenn *Wnd* 0 ist, ruft *GetMessage* Botschaften für jedes Fenster ab, das zur aufrufenden Anwendung gehört.

MsgFilterMin: Bestimmt den Integerwert des niedrigsten abzurufenden Botschaftswerts.

MsgFilterMax: Bestimmt den Integerwert des höchsten abzurufenden Botschaftswerts.

Rückgabewert Der Rückgabewert ist *True*, wenn eine andere Botschaft als *wm_Quit* abgerufen wurde, er ist *False*, wenn die Botschaft *wm_Quit* abgerufen wurde.

Der Rückgabewert wird normalerweise benutzt, um zu entscheiden, ob die Hauptschleife der Anwendung beendet und das Programm verlassen werden soll.

Hinweis

Die Funktionen *PeekMessage* und *GetMessage* geben die Steuerung an andere Anwendungen weiter, wenn Botschaften *wm_Paint* oder *wm_Timer* für andere Tasks vorliegen oder wenn keine Botschaften verfügbar sind. Ein Aufruf einer der Funktionen *GetMessage*, *PeekMessage* und *WaitMessage* ist die einzige Möglichkeit, andere Anwendungen laufen zu lassen. Solange eine Anwendung keine dieser Funktionen aufruft, können keine anderen Anwendungen laufen.

Geben *GetMessage*, *PeekMessage* und *WaitMessage* die Steuerung an andere Anwendungen ab, können sich die Stack- und Datensegmente der die Funktion aufrufenden Anwendung verschieben, um sich dem veränderten Speicherplatzbedarf anderer Anwendungen anzupassen. Wenn die Anwendung Zeiger auf Objekte im Daten- oder Stacksegment (das sind globale oder lokale Variablen) gespeichert hat, können diese Zeiger nach einem Aufruf von *GetMessage*, *PeekMessage* oder *WaitMessage* ungültig werden. Der Parameter *Msg* der aufgerufenen Funktion bleibt in jedem Fall gültig.

GetMessagePos

Syntax

```
function GetMessagePos: LongInt;
```

Beschreibung

Diese Funktion gibt einen Wert zurück, der die Mauszeigerposition in Bildschirmkoordinaten zu dem Zeitpunkt darstellt, zu dem die letzte von der Funktion *GetMessage* abgerufene Botschaft eintraf.

Diese Funktion hat keine Parameter.

Rückgabewert

Der Rückgabewert gibt die x- und y-Koordinaten der Mauszeigerposition zurück. Die x-Koordinate liegt im niederwertigen, die y-Koordinate im höherwertigen Word. Wenn der Rückgabewert einer Variablen zugewiesen ist, kann der Typbezeichner *MakePoint* benutzt werden, um eine *TPoint*-Struktur aus dem Rückgabewert zu erhalten.

Hinweis

Um die aktuelle Position des Mauszeigers anstelle seiner Position beim letzten Eintreffen der Botschaft zu erhalten, kann die Prozedur *GetCursorPos* verwendet werden.

GetMessageTime

Syntax

```
function GetMessageTime: LongInt;
```

Beschreibung

Diese Funktion gibt die Zeit der Erzeugung der letzten von *GetMessage* abgerufenen Botschaft an. Die Zeit liegt als LongInt

vor. Dieser bezeichnet die Zeit in Millisekunden, die vom Booten des Systems bis zu dem Zeitpunkt, an dem die Botschaft erzeugt (in die Warteschlange gestellt) wurde, vergangen ist.

Diese Funktion hat keine Parameter.

Rückgabewert Der Rückgabewert gibt die Zeit der Erzeugung der Botschaft zurück.

Hinweis Man sollte nicht voraussetzen, daß sich der Rückgabewert immer vergrößert. Der Rückgabewert wird auf 0 zurückgesetzt, wenn der Timer-Zählwert den Maximalwert für LongInt-Zahlen überschreitet.

Um das Zeitintervall zwischen Botschaften zu berechnen, subtrahiert man die Zeit der ersten Botschaft von der der zweiten.

GetMetaFile

Syntax
```
function GetMetaFile(FileName: PChar): THandle;
```

Beschreibung Diese Funktion erzeugt ein Handle für das Metafile, das vom Parameter *FileName* benannt wird.

FileName: Zeigt auf den nullterminierten String, der den DOS-Dateinamen des Metafiles angibt. Es wird vorausgesetzt, daß das Metafile existiert.

Rückgabewert Der Rückgabewert bezeichnet ein Metafile, wenn die Funktion erfolgreich war, andernfalls ist er 0.

GetMetaFileBits

Syntax
```
function GetMetaFileBits(MF: THandle): THandle;
```

Beschreibung Diese Funktion gibt ein Handle auf einen globalen Speicherblock zurück, der das angegebene Metafile als eine Ansammlung von Bits enthält.

Der Speicherblock kann benutzt werden, um die Größe des Metafiles zu bestimmen oder um das Metafile als Datei abzuspeichern.

Der Speicherblock sollte nicht verändert werden.

MF: Bezeichnet das Speicher-Metafile.

Rückgabewert Der Rückgabewert bezeichnet den globalen Speicherblock, der das Metafile enthält. Wenn ein Fehler auftritt, ist der Rückgabewert 0.

Hinweis Das als Parameter *MF* benutzte Handle wird ungültig, wenn die Funktion *GetMetaFileBits* zurückkehrt. Aus diesem Grund muß das zurückgegebene globale Speicher-Handle benutzt werden, um sich auf das Metafile zu beziehen.

Von dieser Funktion erzeugte Speicherblocks sind nur für die aufrufende Anwendung bestimmt und werden von anderen Anwendungen nicht mitbenutzt. Diese Blocks werden automatisch gelöscht, wenn die Anwendung beendet wird.

GetModuleFileName

Syntax
```
function GetModuleFileName(Module: THandle; FileName:
PChar; Size: Integer): Integer;
```

Beschreibung Diese Funktion ruft den vollständigen Pfadnamen der ausführbaren Datei ab, aus der das angegebene Modul geladen wurde.

GetModuleFileName kopiert den nullterminierten Dateinamen in den Puffer, auf den der Parameter *FileName* zeigt.

Module: Bezeichnet das Modul oder die Instanz des Moduls.

FileName: Zeigt auf den nullterminierten String, der den Dateinamen empfangen soll.

Size: Bestimmt die maximale Anzahl von Zeichen, die kopiert werden sollen. Wenn der Dateiname länger als die vom Parameter *Size* angegebene Maximalzahl von Zeichen ist, wird er abgeschnitten.

Rückgabewert Der Rückgabewert gibt die tatsächliche Länge des in den Puffer kopierten Strings an.

GetModuleHandle

Syntax
```
function GetModuleHandle(ModuleName: PChar): THandle;
```

Beschreibung Diese Funktion ruft das Modul-Handle des angegebenen Moduls ab.

ModuleName: Zeigt auf eine nullterminierten String, der das Modul bezeichnet.

Rückgabewert Wenn die Funktion erfolgreich ausgeführt werden konnte, bezeichnet der Rückgabewert das Modul, andernfalls wird 0 zurückgegeben.

GetModuleUsage

Syntax
```
function GetModuleUsage(Module: THandle): Integer;
```

Beschreibung Diese Funktion ruft den Referenzzähler eines angegebenen Moduls ab.

Module: Bezeichnet das Modul oder die Instanz des Moduls.

Rückgabewert Der Rückgabewert bezeichnet den Referenzzähler des Moduls.

GetNearestColor

Syntax
```
function GetNearestColor(DC: HDC; Color: TColorRef): TColorRef;
```

Beschreibung Diese Funktion gibt die einer angegebenen virtuellen Farbe ähnlichste virtuelle Farbe zurück, die vom angegebene Gerät dargestellt werden kann.

DC: Bezeichnet den Gerätekontext.

Color: Bestimmt die Farbe, die angeglichen werden soll.

Rückgabewert Der Rückgabewert zeigt einen RGB-Farbwert an, der die Flächenfarbe bezeichnet, die das Gerät darstellen kann und der dem Wert *Color* am nächsten liegt.

GetNearestPaletteIndex

Syntax
```
function GetNearestPaletteIndex(Palette: HPalette; Color: TColorRef): Word;
```

Beschreibung Diese Funktion gibt den Index des Eintrags in einer virtuellen Palette zurück, der einem vorgegebenen Farbwert am ähnlichsten ist.

Palette: Bezeichnet die virtuelle Palette.

Color: Bestimmt die Farbe, die angeglichen werden soll.

Rückgabewert Der Rückgabewert ist der Index des Eintrags in einer virtuellen Palette. Der Eintrag enthält die Farbe, die der angegebenen Farbe am ähnlichsten ist.

GetNextDlgGroupItem

Syntax
```
function GetNextDlgGroupItem(Dlg, Ctl: HWnd; Previous: Bool): HWnd;
```

Beschreibung Diese Funktion sucht in dem vom Parameter *Dlg* bezeichneten Dialogfenster nach dem nächsten (oder vorherigen) Steuerelement innerhalb einer Gruppe von Steuerelementen.

Eine Gruppe von Steuerelementen besteht aus einer oder mehreren Steuerelementen mit dem Stil *ws_Group*.

Dlg: Bezeichnet das durchsuchte Dialogfenster.

Ctl: Bezeichnet das Steuerelement im Dialogfenster, bei dem die Suche beginnt.

Previous: Bestimmt, wie die Funktion die Gruppe von Steuerungen im Dialogfenster absuchen soll. Ist *Previous False*, sucht die Funktion nach dem vorhergehenden Steuerelement in der

Gruppe, ist *Previous True*, sucht die Funktion nach dem nächsten Steuerelement in der Gruppe.

Rückgabewert Der Rückgabewert bezeichnet das nächste oder vorherige Steuerelement in der Gruppe.

Hinweis Ist das aktuelle Element das letzte Element in der Gruppe und ist *Previous False*, gibt die Funktion *GetNextDLgGroupItem* das Fenster-Handle des ersten Elements in der Gruppe zurück.

Ist das aktuelle Element das erste Element in der Gruppe und ist *Previous True*, dann gibt die Funktion *GetNextDlgGroupItem* das Fenster-Handle des letzten Elements in der Gruppe zurück.

GetNextDlgTabItem

Syntax
```
function GetNextDlgTabItem(Dlg, Ctl: HWnd; Previous: Bool): HWnd;
```

Beschreibung Diese Funktion ruft das Handle des ersten Steuerelements ab, das den Stil *ws_TabStop* hat und dem vom Parameter *Ctl* bezeichneten Steuerelement vorangeht oder darauf folgt.

Dlg: Bezeichnet das zu durchsuchende Dialogfenster.

Ctl: Bezeichnet das Steuerelement, das als Startpunkt der Suche benutzt werden soll.

Previous: Bestimmt, wie die Funktion das Dialogfenster absuchen soll. Ist der Parameter *Previous False*, sucht die Funktion das vorhergehende Steuerelement im Dialogfenster, ist er *True*, sucht die Funktion nach dem nächsten Steuerelement im Dialogfenster.

Rückgabewert Der Rückgabewert bezeichnet das vorherige (oder nächste) Steuerelement, das den Stil *ws_TabStop* gesetzt hat.

GetNextWindow

Syntax
```
function GetNextWindow(Wnd: HWnd; Flag: Word): HWnd;
```

Beschreibung Diese Funktion sucht nach einem Handle, das das nächste (oder vorherige) Fenster in der Liste des Fenster-Managers bezeichnet. Die Liste des Fenster-Managers enthält Einträge für alle Fenster höchster Ebene, die ihnen zugehörigen untergeordneten Fenster und die untergeordneten Fenster aller untergeordneten Fenster.

Wenn der Parameter *Wnd* ein Handle zu einem Hauptfenster ist, sucht die Funktion nach dem nächsten (oder vorherigen) Handle zu einem Hauptfenster. Ist *Wnd* ein Handle zu einem untergeordneten Fenster, sucht die Funktion nach

einem Handle zum nächsten (oder vorherigen) untergeordneten Fenster.

Wnd: Bezeichnet das aktuelle Fenster.

Flag: Bestimmt, ob die Funktion ein Handle zum nächsten oder zum vorherigen Fenster zurückgibt. Er kann einen der folgenden Werte annehmen:

Wert	Bedeutung
gw_HWndNext	Die Funktion gibt ein Handle zum nächsten Fenster zurück.
gw_HWndPrev	Die Funktion gibt ein Handle zum vorherigen Fenster zurück.

Rückgabewert Der Rückgabewert bezeichnet das nächste (oder vorherige) Fenster in der Liste des Fenster-Managers.

GetNumTasks

Syntax
```
function GetNumTasks: Word;
```

Beschreibung Diese Funktion gibt die Anzahl von Tasks zurück, die gerade im System ablaufen. Eine Task ist eine einzelne Instanz einer Windows-Anwendung. Diese Funktion hat keine Parameter.

Rückgabewert Der Rückgabewert ist ein Integerwert, der die Anzahl der Tasks bezeichnet, die gerade im System ausgeführt werden.

GetObject

Syntax
```
function GetObject(hObject: THandle; Count: Integer;
lpObject: Pointer): Integer;
```

Beschreibung Diese Funktion füllt einen Puffer mit den virtuellen Daten, die das vom Parameter *hObject* bezeichnete virtuelle Objekt definieren. *GetObject* kopiert die vom Parameter *Count* vorgegebene Anzahl von Datenbytes in den Puffer, auf den der Parameter *lpObject* zeigt. Die Funktion ruft, abhängig vom virtuellen Objekt, Datenstrukturen des Typs *TLogPen*, *TLogBrush*, *TLogFont*, *TBitmap* oder einen Integerwert ab. Der Puffer muß ausreichend groß sein, um die Daten aufnehmen zu können.

Wenn *hObject* ein Bitmap bezeichnet, zeigt die Funktion nur die Formatinformationen über Breite, Höhe und Farbe des Bitmaps an. Die wirklichen Bits müssen mit der Funktion *GetBitmapBits* abgerufen werden. Wenn *hObject* eine virtuelle Palette angibt, ruft sie einen Zwei-Byte-Wert ab, der die Anzahl der Einträge in der Palette enthält. Sie ruft nicht die gesamte Datenstruktur *TLogPalette* ab, die die Palette definiert. Um

Informationen über Paletteneinträge abzurufen, muß eine Anwendung die Funktion *GetPaletteEntries* aufrufen.

hObject: Bezeichnet einen virtuellen Stift, Pinsel, eine virtuelle Schrift oder Palette oder ein virtuelles Bitmap.

Count: Gibt die Anzahl von Bytes an, die in den Puffer kopiert werden sollen.

lpObject: Zeigt auf den Puffer, der die Informationen aufnehmen soll.

Rückgabewert Der Rückgabewert gibt die tatsächliche Anzahl abgerufener Bytes an. Er ist 0, wenn ein Fehler auftritt.

Querverweis *GetBitmapBits, GetPaletteEntries*

GetPaletteEntries

Syntax
```
function GetPaletteEntries(Palette: HPalette;
StartIndex, NumEntries: Word; var PaletteEntries): Word;
```

Beschreibung Diese Funktion ruft einen Bereich von Paletteneinträgen in einer virtuellen Palette ab.

Palette: Bezeichnet die virtuelle Palette.

StartIndex: Bezeichnet den ersten Eintrag in der virtuellen Palette, der abgerufen werden soll.

NumEntries: Gibt die Zahl der Eintrage in der virtuellen Palette an, die abgerufen werden sollen.

PaletteEntries: Zeigt auf ein Array von *TPaletteEntry*-Datenstrukturen, das die Paletteneinträge aufnehmen sollen. Das Array muß mindestens soviele Datenstrukturen enthalten, wie vom Parameter *NumEntries* vorgegeben wird.

Rückgabewert Der Rückgabewert ist die Anzahl von Einträgen, die aus der virtuellen Palette abgerufen wurden, im Fall eines Fehlers 0.

GetParent

Syntax
```
function GetParent(Wnd: HWnd): HWnd;
```

Beschreibung Diese Funktion ruft das Fenster-Handle des übergeordneten Fensters des angegebenen Fensters ab.

Wnd: Bezeichnet das Fenster, für welches das Handle seines übergeordneten Fensters abgerufen werden soll.

Rückgabewert Der Rückgabewert bezeichnet das übergeordnete Fenster. Wenn das Fenster kein übergeordnetes Fenster hat, wird 0 zurückgegeben.

GetPixel

Syntax
```
function GetPixel(DC: HDC; X, Y: Integer): TColorRef;
```

Beschreibung Diese Funktion ruft den RGB-Farbwert des Pixels an dem durch die Parameter *X* und *Y* bezeichneten Punkt ab. Der Punkt muß in der Clipping-Region liegen, ist dies nicht der Fall, wird die Funktion ignoriert.

DC: Bezeichnet den Gerätekontext.

X: Bestimmt die virtuelle x-Koordinate des zu überprüfenden Punktes.

Y: Bestimmt die virtuelle y-Koordinate des zu überprüfenden Punktes.

Rückgabewert Der Rückgabewert bezeichnet einen RGB-Farbwert für die Farbe des angegebenen Punktes. Er ist -1, wenn die Koordinaten keinen Punkt in der Clipping-Region bezeichnen.

Hinweis Nicht alle Geräte unterstützen die Funktion *GetPixel*. Vergleichen Sie hierzu die Beschreibung der Funktion *GetDeviceCaps*.

GetPolyFillMode

Syntax
```
function GetPolyFillMode(DC: HDC): Integer;
```

Beschreibung Diese Funktion ruft den aktuellen Polygonfüllmodus ab.

DC: Bezeichnet den Gerätekontext.

Rückgabewert Der Rückgabewert gibt den Polygonfüllmodus an. Er kann einen der folgenden Werte annehmen:

Wert	Bedeutung
Alternate	Füllt verschiedene Regionen.
Winding	Füllt alle Regionen.

Weitere Informationen finden Sie in der Beschreibung der Funktion *SetPolyFillMode*.

GetPriorityClipboardFormat

Syntax
```
function GetPriorityClipboardFormat(var PriorityList;
Count: Integer): Integer;
```

Beschreibung Diese Funktion gibt das erste Zwischenablageformat einer Liste zurück, für das Daten existieren.

PriorityList: Zeigt auf ein Integer-Array, das eine Liste von Zwischenablageformaten enthält, die nach ihrer Priorität

geordnet sind. Es handelt sich dabei um die Konstanten, die in Kapitel 1 »Windows-Stile und -Konstanten« im Abschnitt »cf_Xxxx« beschrieben werden.

Count: Bestimmt die Anzahl der Einträge in *PriorityList*. Dieser Wert darf nicht größer als die tatsächliche Zahl von Einträgen in der Liste sein.

Rückgabewert Der Rückgabewert ist das Zwischenablageformat in der Liste mit der höchsten Priorität, für das Daten vorhanden sind. Sind in der Zwischenablage keine Daten vorhanden, gibt die Funktion 0 zurück, sind Daten vorhanden, die zu keinem Format in der Liste passen, ist der Rückgabewert -1.

GetPrivateProfileInt

Syntax
```
function GetPrivateProfileInt(ApplicationName, KeyName:
PChar; Default: Integer; FileName: PChar): Integer;
```

Beschreibung Diese Funktion ruft den Wert einer Integerschlüsselanweisung aus der bezeichneten Initialisierungsdatei ab.

GetPrivateProfileInt sucht die Datei nach einer durch den Parameter *KeyName* angegebenen Schlüsselanweisung ab und überprüft dann ihre Übereinstimmung mit der Schlüsselanweisung unter dem Programmkopf der durch *ApplicationName* bezeichneten Anwendung.

Ein Integereintrag in der Initialisierungsdatei muß folgende Form haben:

```
[Anwendungsname]
Schlisselname = Wert
.
.
.
```

ApplicationName: Zeigt auf den Namen einer Windows-Anwendung, die in der Initialisierungsdatei erscheint.

KeyName: Zeigt auf den Schlüsselnamen, der in der Initialisierungsdatei erscheint.

Default: Bestimmt den Standardwert der angegebenen Schlüsselanweisung, wenn die Schlüsselanweisung nicht in der Initialisierungsdatei gefunden werden kann.

FileName: Zeigt auf einen nullterminierten String, der die Initialisierungsdatei bezeichnet. Enthält *FileName* keine Pfadangabe für die Datei, sucht Windows nach der Datei im Windows-Verzeichnis.

Rückgabewert Der Rückgabewert ist 0, wenn der zum angegebenen Schlüsselnamen gehörende Wert kein Integerwert oder ein negativer

Integerwert ist. Besteht der Wert, der zum Schlüsselnamen gehört, aus Ziffern, auf die nichtnumerische Zeichen folgen, gibt die Funktion den Wert der Ziffern zurück. Wird zum Beispiel auf den Eintrag *KeyName=102abc* zugegriffen, gibt die Funktion *102* zurück, wird die Schlüsselanweisung nicht gefunden, ist der Rückgabewert der Standardwert *Default*.

Hinweis

Die Funktion *GetPrivateProfileInt* berücksichtigt die Schreibweise nicht. Aus diesem Grund können die Strings in *ApplicationName* und *KeyName* aus einer beliebigen Kombination von großen und kleinen Buchstaben bestehen.

GetPrivateProfileString

Syntax

```
function GetPrivateProfileString(ApplicationName,
KeyName, Default, ReturnedString: PChar; Size: Integer;
FileName: PChar): Integer;
```

Beschreibung

Diese Funktion kopiert einen String aus der angegebenen Initialisierungsdatei in einen Puffer, auf den der Parameter *ReturnedString* zeigt. Sie durchsucht die Datei nach einer Schlüsselanweisung, die mit der vom Parameter *KeyName* vorgegebenen Bezeichnung unter dem Programmkopf der vom Parameter *ApplicationName* vorgegebenen Anwendung übereinstimmt. Wird diese gefunden, wird der zugehörende String in den Puffer kopiert. Ist die Schlüsselanweisung nicht vorhanden, wird die durch den Parameter *Default* angegebene Standardzeichenkette in den Puffer kopiert. Ein Stringeintrag in der Initialisierungsdatei muß folgende Form haben:

```
[Anwendungsname]
Schlisselname = string
.
.
.
```

Ist *KeyName* 0, listet *GetPrivateProfileString* alle Schlüsselnamen auf, die *ApplicationName* zugeordnet sind, indem sie die Stelle, auf die *ReturnedString* zeigt, mit einer Liste von Schlüsselnamen füllt (nicht mit Werten). Jede Schlüsselanweisung in der Liste wird mit 0 abgeschlossen.

ApplicationName: Zeigt auf den Namen einer Windows-Anwendung, der in der Initialisierungsdatei erscheint.

KeyName: Zeigt auf einen Schlüsselnamen, der in der Initialisierungsdatei erscheint.

Default: Bestimmt den Standardwert der bezeichneten Schlüsselanweisung, wenn die Schlüsselanweisung in der Initialisierungsdatei nicht gefunden werden kann.

ReturnedString: Zeigt auf den Puffer, der den String aufnehmen soll.

Size: Gibt die maximale Anzahl von Zeichen an (einschließlich Nullzeichen), die in den Puffer kopiert werden sollen.

FileName: Zeigt auf einen String, der die Initialisierungsdatei benennt. Enthält *FileName* keine Pfadangabe für die Datei, sucht Windows die Datei im Windows-Verzeichnis.

Rückgabewert Der Rückgabewert gibt die Anzahl der in den vom Parameter *ReturnedString* bezeichneten Puffer kopierten Zeichen an. Die abschließenden Nullzeichen werden dabei nicht berücksichtigt. Wenn *KeyName* nicht 0 und der Puffer nicht groß genug ist, um den ganzen String aufzunehmen, ist der Rückgabewert gleich der vom Parameter *Size* vorgegebenen Länge. Ist *KeyName* 0 und der Puffer nicht groß genug, um den ganzen String aufzunehmen, ist der Rückgabewert um 2 kleiner als die vom Parameter *Size* vorgegebene Länge.

Hinweis *GetPrivateProfileString* berücksichtigt die Schreibweise nicht, so daß die Strings in *ApplicationName* und *KeyName* aus einer beliebigen Kombination von großen und kleinen Buchstaben bestehen können.

GetProcAddress

Syntax
```
function GetProcAddress(Module: THandle; ProcName:
PChar): TFarProc;
```

Beschreibung Diese Funktion ruft die Speicheradresse der Funktion ab, auf deren Bezeichnung der Parameter *ProcName* zeigt.

Module: Bezeichnet das Bibliotheksmodul, das die Funktion enthält.

ProcName: Zeigt auf die Funktionsbezeichnung oder enthält die Ordnungszahl der Funktion. Wenn es eine Ordnungszahl ist, so muß diese im niederwertigen und 0 im höherwertigen Word liegen. *ProcName* muß ein nullterminierter String sein.

Rückgabewert Bei erfolgreicher Ausführung zeigt der Rückgabewert auf den Einsprungspunkt der Funktion, andernfalls ist er 0.

Ist der Parameter *ProcName* eine Ordnungszahl und existiert keine Funktion mit der angegebenen Ordnungszahl in dem Modul, kann *GetProcAddress* trotzdem noch einen von 0 verschiedenen Wert zurückgeben. Man sollte daher in den Fällen, in denen die Funktion eventuell nicht existiert, die Funktion mit der Bezeichnung anstatt mit der Ordnungszahl benennen.

Hinweis *GetProcAddress* sollte nur dazu verwendet werden, Adressen von exportierten Funktionen abzurufen, die zu Bibliotheks-

modulen gehören. Die Funktion *MakeProcInstance* kann dazu benutzt werden, auf Funktionen innerhalb unterschiedlicher Instanzen des aktuellen Moduls zuzugreifen.

GetProfileInt

Syntax

```
function GetProfileInt(AppName, KeyName: PChar;
Default: Integer): Word;
```

Beschreibung

Diese Funktion ruft den Wert einer Integerschlüsselanweisung aus der Windows-Initialisierungsdatei WIN.INI ab.

GetProfileInt durchsucht die Datei WIN.INI nach einer Schlüsselanweisung unter dem durch den Parameter *AppName* bezeichneten Anwendungsprogrammkopf, die mit der durch den Parameter *KeyName* benannten Bezeichnung übereinstimmt.

Ein Integereintrag in WIN.INI muß folgende Form haben:

```
[Anwendungsname]
Schlisselname = Wert
.
.
.
```

AppName: Zeigt auf die Bezeichnung einer Windows-Anwendung, die in der Windows-Initialisierungsdatei erscheint.

KeyName: Zeigt auf einen Schlüsselnamen, der in der Windows-Initialisierungsdatei erscheint.

Default: Bestimmt den Standardwert für die gegebene Schlüsselanweisung, wenn die Schlüsselanweisung in der Windows-Initialisierungsdatei nicht gefunden werden kann.

Rückgabewert

Der Rückgabewert ist 0, wenn der zur angegebenen Schlüsselbezeichnung gehörende Wert kein Integerwert oder ein negativer Integerwert ist. Besteht der Wert, der zum Schlüsselnamen gehört, aus Ziffern, auf die nichtnumerische Zeichen folgen, gibt die Funktion den Wert der Ziffern zurück.

Wenn z.B. auf den Eintrag *KeyName=102abc* zugegriffen wird, gibt die Funktion *102* zurück, wird die Schlüsselanweisung nicht gefunden, gibt sie den Standardwert *Default* zurück.

GetProfileString

Syntax

```
function GetProfileString(AppName, KeyName, Default,
ReturnedString: PChar; Size: Integer): Integer;
```

Beschreibung

Diese Funktion kopiert einen String aus der Windows-Initialisierungsdatei WIN.INI in den Puffer, auf den der Parameter *ReturnedString* zeigt.

GetProfileString durchsucht WIN.INI nach einer Schlüsselanweisung unter dem durch den Parameter *AppName* bezeichneten Anwendungsprogrammkopf, die mit der durch den Parameter *KeyName* benannten Bezeichnung übereinstimmt.

Wenn die Schlüsselanweisung gefunden wird, dann wird der entsprechende String in den Puffer kopiert, andernfalls wird die im Parameter *Default* angegebene Voreinstellung kopiert.

Ein Stringeintrag in WIN.INI muß die folgende Form haben:

```
[Anwendungsname]
Schlisselname = Wert
.
.
.
```

Ist *KeyName* 0, listet die Funktion *GetProfileString* alle Schlüsselnamen auf, die *AppName* zugeordnet sind, indem sie die Position, auf die der Parameter *ReturnedString* zeigt, mit einer Liste von Schlüsselnamen (nicht Werten) füllt. Jeder Schlüsselname in der Liste ist mit einem Nullzeichen abgeschlossen.

AppName: Zeigt auf einen nullterminierten String, der die Anwendung bezeichnet.

KeyName: Zeigt auf einen nullterminierten String, der eine Schlüsselanweisung bezeichnet.

Default: Bestimmt den voreingestellten Wert für die angegebene Schlüsselanweisung, wenn die Schlüsselanweisung in der Initialisierungsdatei nicht gefunden werden kann.

ReturnedString: Zeigt auf den Puffer, der die String empfängt.

Size: Bestimmt die Anzahl der Zeichen (einschließlich des letzten Nullzeichens), die in den Puffer kopiert werden.

Rückgabewert

Der Rückgabewert gibt die Anzahl der in den vom Parameter *ReturnedString* bezeichneten Puffer kopierten Zeichen an, ohne Berücksichtigung des abschließenden Nullzeichens.

Wenn der Puffer nicht groß genug ist, um den gesamten String aufzunehmen, und *KeyName* ungleich 0 ist, ist der Rückgabewert gleich der vom Parameter *Size* angegebenen Länge.

Wenn der Puffer nicht groß genug ist, um den gesamten String aufzunehmen, und *KeyName* 0 ist, ist der Rückgabewert um 2 kleiner als die vom Parameter *Size* vorgegebene Länge.

Hinweis

GetProfileString läßt die Schreibweise unberücksichtigt, so daß die Strings in *AppName* und *KeyName* in einer beliebigen Kombination von Groß- und Kleinschreibung eingegeben werden können.

GetProp

Syntax
```
function GetProp(Wnd: HWnd; Str: PChar): THandle;
```

Beschreibung Diese Funktion ruft ein Daten-Handle aus der Eigenschaftsliste des angegebenen Fensters ab. Der String, auf den der Parameter *Str* zeigt, bezeichnet das Handle, das abgerufen werden soll.

Es wird vorausgesetzt, daß der String und das Handle durch die Funktion *SetProp* zu der Eigenschaftsliste hinzugefügt wurden.

Wnd: Bezeichnet das Fenster, dessen Eigenschaftsliste durchsucht werden soll.

Str: Zeigt auf einen nullterminierten String oder ein Atom, das einen String bezeichnet. Wenn ein Atom angegeben ist, muß dies vorher mit der Funktion *AddAtom* erzeugt worden sein. Das Atom, ein 16-Bit-Wert, muß im niederwertigen Word des Parameters *Str* positioniert werden, das höherwertige Word muß auf 0 gesetzt werden.

Rückgabewert Wenn die Eigenschaftsliste den angegebenen String enthält, bezeichnet der Rückgabewert das zugehörende Daten-Handle, andernfalls ist er 0.

Hinweis Der von der Funktion *GetProp* abgerufene String kann jeder für die Anwendung verwendbare 16-Bit-Wert sein.

Querverweis *AddAtom*

GetRgnBox

Syntax
```
function GetRgnBox(Rgn: HRgn; var Rect: TRect): Integer;
```

Beschreibung Diese Funktion ruft die Koordinaten des umgrenzenden Rechtecks der durch den Parameter *Rgn* bezeichneten Region ab.

Rgn: Bezeichnet die Region.

Rect: Eine *TRect*-Datenstruktur, die die Koordinaten des umgrenzenden Rechtecks aufnehmen soll.

Rückgabewert Der Rückgabewert gibt den Typ der Region an. Er kann einen der folgenden Werte annehmen.

Wert	Bedeutung
ComplexRegion	Region hat überlappende Grenzen.
NullRegion	Region ist leer.
SimpleRegion	Region hat keine überlappenden Grenzen.

Der Rückgabewert ist 0, wenn der Parameter *Rgn* keine gültige Region bezeichnet.

GetROP2

Syntax
```
function GetROP2(DC: HDC): Integer;
```

Beschreibung Diese Funktion ruft den aktuellen Zeichenmodus ab.

Der Zeichenmodus legt fest, wie der Stift oder die innere Farbe und die bereits auf der Bildschirmfläche befindliche Farbe kombiniert werden sollen, um eine neue Farbe zu ergeben.

DC: Bezeichnet den Gerätekontext für ein rasterorientiertes Gerät.

Rückgabewert Der Rückgabewert gibt den Zeichenmodus an. Der Abschnitt »r2_Xxxx« in Kapitel 1 »Windows-Stile und -Konstanten« enthält die möglichen Werte. Zusätzliche Informationen finden Sie auch unter der Beschreibung der Funktion *SetROP2*.

Querverweis *SetROP2*

GetRValue

Syntax
```
function GetRValue(RGB: LongInt): Byte;
```

Beschreibung Diese Inline-Funktion entnimmt einem RGB-Farbwert den Wert für Rot.

RGB: Bezeichnet ein rotes, ein blaues und ein grünes Farbfeld. Jedes der Felder gibt die Intensität der angegebenen Farbe vor.

Rückgabewert Der Rückgabewert bezeichnet ein Byte, das den roten Farbwert des Parameters *RGB* enthält.

Hinweis Der Wert 255 entspricht der maximalen, 0 der minimalen Intensität für ein einzelnes Byte.

GetScrollPos

Syntax
```
function GetScrollPos(Wnd: HWnd; Bar: Integer): Integer;
```

Beschreibung Diese Funktion ruft die aktuelle Position eines Bildlaufleistensymbols ab. Die aktuelle Position ist ein relativer Wert, der vom aktuellen Bildlaufbereich abhängt. Geht der Bildlaufbereich zum Beispiel von 0 bis 100 und das Symbol ist in der Mitte der Leiste, ist die aktuelle Position 50.

Wnd: Bezeichnet – abhängig vom Wert des Parameters *Bar* – ein Fenster mit Standardbildlaufleisten oder eine selbständige Bildlaufleiste.

Bar: Gibt die Bildlaufleiste an, die überprüft werden soll. Er kann einen der folgenden Werte annehmen:

Wert	Bedeutung
sb_Ctl	Ruft die Position einer selbständigen Bildlaufleiste ab. In diesem Fall muß der Parameter *Wnd* das Fenster-Handle einer Bildlaufleiste sein.
sb_Horz	Ruft die Position der horizontalen Bildlaufleiste eines Fensters ab.
sb_Vert	Ruft die Position der vertikalen Bildlaufleiste eines Fensters ab.

Rückgabewert Gibt die aktuelle Position des Bildlaufleistensymbols an.

GetScrollRange

Syntax
```
procedure GetScrollRange(Wnd: HWnd; Bar: Integer; var
MinPos, MaxPos: Integer);
```

Beschreibung Diese Funktion kopiert die für die angegebene Bildlaufleiste aktuellen maximalen und minimalen Positionen des Bildlaufsymbols in die Parameter *MinPos* und *MaxPos*. Wenn das angegebene Fenster keine Standardbildlaufleisten hat oder keine selbständige Bildlaufleiste ist, kopiert *GetScrollRange* 0 in *MinPos* und *MaxPos*.

Wnd: Bezeichnet ein Fenster mit Standardbildlaufleisten oder eine selbständige Bildlaufleiste, abhängig vom Wert des Parameters *Bar*.

Bar: Bestimmt einen Integerwert, der die abzurufende Bildlaufleiste bezeichnet. Möglich ist einer der folgenden Werte:

Wert	Bedeutung
sb_Ctl	Ruft die Position einer selbständigen Bildlaufleiste ab, in diesem Fall muß der Parameter *Wnd* das Handle einer Bildlaufleiste sein.
sb_Horz	Ruft die Position der horizontalen Bildlaufleiste eines Fensters ab.
sb_Vert	Ruft die Position der vertikalen Bildlaufleiste eines Fensters ab.

MinPos: Integervariable, die die minimale Position aufnehmen soll.

MaxPos: Integervariable, die die maximale Position aufnehmen soll.

Hinweis Der voreingestellte Bereich für eine Standardbildlaufleiste geht von 0 bis 100. Der voreingestellte Bereich für eine selbständige Bildlaufleiste ist leer (beide Werte sind 0).

GetStockObject

Syntax

```
function GetStockObject(Index: Integer): THandle;
```

Beschreibung

Diese Funktion ruft ein Handle zu einem der vordefinierten Stifte, Pinsel oder Schriften ab.

Index: Bestimmt den Typ des gewünschten vordefinierten Objekts. Dieser Parameter kann einen der folgenden Werte annehmen:

Wert	Bedeutung
Black_Brush	Schwarzer Pinsel.
DKGray_Brush	Dunkelgrauer Pinsel.
Gray_Brush	Grauer Pinsel.
Hollow_Brush	Hohler Pinsel.
LTGray_Brush	Hellgrauer Pinsel.
Null_Brush	Neutraler Pinsel.
White_Brush	Weißer Pinsel.
Black_Pen	Schwarzer Stift.
Null_Pen	Null-Stift.
White_Pen	Weißer Stift.
ANSI_Fixed_Font	ANSI-Systemschrift mit festen Zeichenabständen.
ANSI_Var_Font	ANSI-Systemschrift mit variablen Zeichenabständen.
Device_Default_Font	Geräteabhängige Schrift.
OEM_Fixed_Font	OEM-Schrift mit festen Zeichenabständen.
System_Font	Die Systemschrift. Standardmäßig benutzt Windows die Systemschrift, um Menüs, Dialogfenster und anderen Text darzustellen. In den Windows-Versionen 3.0 und höher ist die Systemschrift eine Proportionalschrift, frühere Windows-Versionen benutzen eine Systemschrift mit festem Zeichenabstand.
System_Fixed_Font	Die Systemschrift mit festem Zeichenabstand, die in früheren Windows-Versionen benutzt wurde. Dieses Objekt ist nur aus Gründen der Kompatibilität verfügbar.
Default_Palette	Standardfarbpalette. Diese Farbpalette besteht aus den 20 statischen Farben, die immer in der Systempalette präsent sind, um Farben in der virtuellen Palette von Hintergrundfenstern anzugleichen.

Rückgabewert

Wenn die Funktion erfolgreich ausgeführt werden konnte, bezeichnet der Rückgabewert das gewünschte virtuelle Objekt, andernfalls ist er 0.

Hinweis Die Objekte *DKGray_Brush*, *Gray_Brush* und *LTGray_Brush* sollten nicht als Hintergrundpinsel oder für einen anderen Zweck in einem Fenster verwendet werden, dessen Klasse nicht die Stile *cs_HRedraw* und *cs_VRedraw* angibt. Die Verwendung eines grauen vordefinierten Pinsels in solchen Fenstern kann zu Fehlausrichtungen von Pinselmustern führen, wenn ein Fenster verschoben oder in der Größe neu festgelegt wurde. Die Ursprünge vordefinierter Pinsel können nicht eingestellt werden (vgl. die Beschreibung der Funktion *SetBrushOrg*).

GetStretchBitMode

Syntax
```
function GetStretchBltMode(DC: HDC): Integer;
```

Beschreibung Diese Funktion ruft den aktuellen Streckmodus ab. Dieser definiert, wie Informationen zu Bitmaps hinzugefügt oder daraus gelöscht werden, die mit der Funktion *StretchBlt* erweitert oder komprimiert werden.

DC: Bezeichnet den Gerätekontext.

Rückgabewert Der Rückgabewert gibt den aktuellen Streckmodus an. Er kann *WhiteOnBlack*, *BlackOnWhite* oder *ColorOnColor* sein (siehe Abschnitt »Streckmodi« in Kapitel 1 »Windows-Stile und -Konstanten«.

Querverweis *SetStretchBltMode*

GetSubMenu

Syntax
```
function GetSubMenu(Menu: HMenu; Pos: Integer): HMenu;
```

Beschreibung Diese Funktion ruft das Menü-Handle eines Pop-Up-Menüs ab.

Menu: Bezeichnet das Menü.

Pos: Bezeichnet die Position des Pop-Up-Menüs im angegebenen Menü. Positionswerte beginnen bei 0 für die erste Menüoption. Die Integer-ID des Pop-Up-Menüs kann in dieser Funktion nicht verwendet werden.

Rückgabewert Der Rückgabewert bezeichnet das gegebene Pop-Up-Menü. Er ist 0, wenn an der angegebenen Position kein Pop-Up-Menü existiert.

GetSysColor

Syntax
```
function GetSysColor(Index: Integer): TColorRef;
```

Beschreibung Diese Funktion ruft die aktuelle Farbe des durch *Index* bezeichneten Bildschirmelements ab. Bildschirmelemente sind die

verschiedenen Teile eines Fensters und des Windows-Bildschirms, die auf dem Systembildschirm erscheinen.

Index: Bestimmt das Bildschirmelement, dessen Farbe abgerufen werden soll. Eine Liste der Indexwerte finden Sie in Kapitel 1 »Windows-Stile und -Konstanten« im Abschnitt »color_-Xxxx« und bei der Beschreibung der Funktion *SetSysColors*.

Rückgabewert Der Rückgabewert gibt einen RGB-Farbwert an, der die Farbe des angegebenen Elements bezeichnet.

Hinweis Systemfarben für Monochrombildschirme werden normalerweise als unterschiedliche Grauschattierungen interpretiert.

Querverweis *SetSysColor*

GetSysModalWindow

Syntax
```
function GetSysModalWindow: HWnd;
```

Beschreibung Diese Funktion gibt das Handle eines systemmodalen Fensters zurück, wenn ein solches vorhanden ist.

Diese Funktion hat keine Parameter.

Rückgabewert Der Rückgabewert bezeichnet das systemmodale Fenster, falls ein solches vorhanden ist, wenn nicht, dann ist der Rückgabewert 0.

GetSystemDirectory

Syntax
```
procedure GetSystemDirectory(Buffer: PChar; Size: Word);
```

Beschreibung Diese Funktion ermittelt den Pfadnamen des Windows-Systemunterverzeichnisses. Das Systemunterverzeichnis enthält Dateien wie Bibliotheken, Treiber und Schriftdateien.

Buffer: Zeigt auf den Puffer, der den nullterminierten String aufnehmen soll, die den Pfadnamen enthält.

Size: Bestimmt die maximale Größe des Puffers in Bytes. Dieser Wert sollte mindestens auf 144 gesetzt werden, um im Puffer genügend Platz für den Pfadnamen zu schaffen.

Rückgabewert Der Rückgabewert ist die Länge des in den Parameter *Buffer* kopierten Strings, wobei das abschließende Nullzeichen nicht mitgezählt wird.

Ist der Rückgabewert größer als *Size*, ist er die Größe des Puffers, die erforderlich ist, um den Pfadnamen aufzunehmen. Der Rückgabewert ist 0, wenn die Funktion fehlschlägt.

Hinweis Der von dieser Funktion abgerufene Pfadname endet nicht mit einem Backslash (\), außer wenn das Systemverzeichnis das Hauptverzeichnis des Datenträgers ist.

Wenn das Systemverzeichnis zum Beispiel den Namen *WINDOWS\SYSTEM* in Laufwerk C: hat, lautet der von der Funktion abgerufene Pfadname des Systemunterverzeichnisses *C:\WINDOWS\SYSTEM*.

GetSystemMenu

Syntax

```
function GetSystemMenu(Wnd: HWnd; bRevert: Bool): HMenu;
```

Beschreibung

Diese Funktion erlaubt es der Anwendung, auf das Systemmenü zuzugreifen, um dieses zu kopieren und/oder zu verändern.

Wnd: Bezeichnet das Fenster, das eine Kopie des Systemmenüs aufnehmen soll.

bRevert: Gibt die auszuführende Aktion an. Ist *bRevert False*, gibt *GetSystemMenu* ein Handle einer Kopie des aktuell verwendeten Systemmenüs zurück. Diese Kopie ist anfangs mit dem Systemmenü identisch, kann aber verändert werden.

Ist *bRevert True*, entfernt *GetSystemMenu* die möglicherweise veränderte Kopie des Systemmenüs, die zu dem angegebenen Fenster gehört, und gibt ein Handle zu der ursprünglichen, unveränderten Version des Systemmenüs zurück.

Rückgabewert

Wenn *bRevert* ungleich 0 ist und das Systemmneü verändert wurde, bezeichnet der Rückgabewert das Systemmenü.

Wenn *bRevert True* ist und das Systemmenü nicht verändert wurde, ist der Rückgabewert 0, ist *bRevert False*, bezeichnet der Rückgabewert eine Kopie des Systemmenüs.

Hinweis

Jedes Fenster, das nicht *GetSystemMenu* benutzt, um eine eigene Kopie des Systemmenüs zu erstellen, erhält ein Standardsystemmenü.

Das von *GetSystemMenu* zurückgegebene Handle kann mit den Funktionen *AppendMenu*, *InsertMenu* oder *ModifyMenu* verwendet werden, um das Systemmenü zu ändern. Das Systemmenü enthält anfänglich Optionen, die durch verschiedene ID-Werte – wie *sc_Close*, *sc_Move* und *sc_Size* – bezeichnet werden.

Menüoptionen im Systemmenü senden *wm_SysCommand*-Botschaften. Alle vordefinierten Systemmenüoptionen haben ID-Nummern, die größer als $F000 sind. Fügt eine Anwendung Befehle zum Systemmenü hinzu, sollte sie ID-Nummern kleiner als $F000 verwenden.

Windows versetzt, abhängig von der Situation, Optionen im Standardsystemmenü automatisch in den nicht-aktivierbaren Zustand. Die Anwendung kann durch Reaktion auf die Botschaft *wm_InitMenu* ihre eigene Auswahl oder ihre eigene Versetzung in den nicht aktivierbaren Zustand herstellen.

Diese Botschaft wird vor der Darstellung eines Menüs übergeben.

Querverweis *AppendMenu, InsertMenu, ModifyMenu*

GetSystemMetrics

Syntax
```
function GetSystemMetrics(Index: Integer): Integer;
```

Beschreibung Diese Funktion ruft die Systemmaße ab. Das sind die Breiten und Höhen verschiedener Bildschirmelemente des Windows-Bildschirms. *GetSystemMetrics* kann auch Flags zurückgeben, die anzeigen, ob die aktuelle Version eine Debugger-Version ist, ob eine Maus angeschlossen ist, oder ob die Bedeutung der linken und rechten Maustaste ausgetauscht worden ist.

Index: Gibt die Systemmaßeinheit an, die abgerufen werden soll. Alle Maßeinheiten werden in Pixeln angegeben.

Die Systemmaßeinheit ist einer der im folgenden Hinweis beschriebenen Werte.

Rückgabewert Der Rückgabewert gibt die angeforderten Systemmaßeinheiten an.

Hinweis Systemmaße hängen vom Systembildschirm ab und können von Bildschirm zu Bildschirm schwanken. Die Werte für den Parameter *Index* sind in der folgenden Übersicht kurz beschrieben:

Index	Bedeutung
sm_CXScreen	Breite des Bildschirms.
sm_CYScreen	Höhe des Bildschirms.
sm_CXFrame	Breite des Fensterrahmens, einstellbar.
sm_CYFrame	Höhe des Fensterrahmens, einstellbar.
sm_CXVScroll	Breite des Pfeilbitmaps auf der vertikalen Bildlaufleiste.
sm_CYVScroll	Höhe des Pfeilbitmaps auf der vertikalen Bildlaufleiste.
sm_CXHScroll	Breite des Pfeilbitmaps auf der horizontalen Bildlaufleiste.
sm_CYHScroll	Höhe des Pfeilbitmaps auf der horizontalen Bildlaufleiste.
sm_CYCaption	Höhe der Titelzeile eines Fensters.
sm_CXBorder	Breite des Fensterrahmens, nicht einstellbar.
sm_CYBorder	Höhe des Fensterrahmens, nicht einstellbar.
sm_CXDlgFrame	Breite des Rahmens, wenn das Fenster den Stil *ws_DlgFrame* hat.
sm_CYDlgFrame	Höhe des Rahmens, wenn das Fenster den Stil *ws_DlgFrame* hat.

Index	Bedeutung
sm_CXHThumb	Breite des Bildlaufsymbols auf der horizontalen Bildlaufleiste.
sm_CYVThumb	Höhe des Bildlaufsymbols auf der vertikalen Bildlaufleiste.
sm_CXIcon	Breite eines Symbols.
sm_CYIcon	Höhe des Symbols.
sm_CXCursor	Breite des Mauszeigers.
sm_CYCursor	Höhe des Mauszeigers.
sm_CYMenu	Höhe einer einzeiligen Menüleiste.
sm_CXFullScreen	Breite eines Fenster-Client-Bereiches für eine Vollbildanwendung.
sm_CYFullScreen	Höhe des Fenster-Client-Bereiches für eine Vollbildanwendung (gleich der Höhe des Bildschirms minus der Höhe der Titelzeile).
sm_CYKanjiWindow	Höhe des Kanji-Fensters.
sm_CXMinTrack	Minimale Tracking-Breite des Fensters.
sm_CYMinTrack	Minimale Tracking-Höhe des Fensters.
sm_CXMin	Minimale Breite des Fensters.
sm_CYMin	Minimale Höhe des Fensters.
sm_CXSize	Breite der Bitmaps, die in der Titelzeile enthalten sind.
sm_CYSize	Höhe der Bitmaps, die in der Titelzeile enthalten sind.
sm_MousePresent	Ungleich 0, wenn eine Maus installiert ist.
sm_Debug	Ungleich 0, wenn die Windows-Debugger-Version vorliegt.
sm_SwapButton	Ungleich 0, wenn die linke und rechte Maustaste vertauscht wurden.

GetSystemPaletteEntries

Syntax

```
function GetSystemPaletteEntries(DC: HDC; StartIndex,
NumEntries: Word; var PaletteEntries): Word;
```

Beschreibung

Diese Funktion ruft einen Bereich von Paletteneinträgen aus der Systempalette ab.

DC: Bezeichnet den Gerätekontext.

StartIndex: Bestimmt den ersten Eintrag in der Systempalette, der abgerufen werden soll.

NumEntries: Bestimmt die Anzahl von Einträgen in der Systempalette, die abgerufen werden sollen.

PaletteEntries: Ein Array von *TPaletteEntry*-Datenstrukturen, das die Paletteneinträge aufnehmen soll. Das Array muß min-

destens soviele Datenstrukturen, wie im Parameter *NumEntries* angegeben ist, enthalten.

Rückgabewert Der Rückgabewert ist die Anzahl von Einträgen, die aus der Systempalette abgerufen wurden. Er ist 0, wenn die Funktion fehlschlägt.

GetSystemPaletteUse

Syntax
```
function GetSystemPaletteUse(DC: HDC): Word;
```

Beschreibung Diese Funktion ermittelt, ob eine Anwendung auf die ganze Systempalette zugreifen kann. Standardmäßig enthält die Systempalette 20 statische Farben, die nicht verändert werden, wenn eine Anwendung ihre virtuelle Palette realisiert. Eine Anwendung kann durch Aufruf der Funktion *SetSystemPaletteUse* Zugriff auf die meisten dieser Farben erlangen.

Der vom Parameter *DC* bezeichnete Gerätekontext muß sich auf ein Gerät beziehen, das Farbpaletten unterstützt.

DC: Bezeichnet den Gerätekontext.

Rückgabewert Der Rückgabewert bestimmt die aktuelle Verwendung der Systempalette. Er hat einen der folgenden Werte:

Wert	Bedeutung
Syspal_NoStatic	Die Systempalette enthält außer schwarz und weiß keine statischen Farben.
Syspal_Static	Die Systempalette enthält statische Farben, die sich nicht ändern, wenn eine Anwendung ihre virtuelle Palette realisiert.

Querverweis *SetSystemPaletteUse*

GetTabbedTextExtent

Syntax
```
function GetTabbedTextExtent(DC: HDC; Str: PChar;
Count, TabPositions: Integer; var TabStopPositions):
LongInt;
```

Beschreibung Diese Funktion berechnet Breite und Höhe der Textzeile, auf die der Parameter *Str* zeigt. Enthält der String ein oder mehrere Tabulatorzeichen, richtet sich die Breite des Strings nach den Tabulatorpositionen, die vom Parameter *TabStopPositions* vorgegeben sind. Die Funktion *GetTabbedTextExtent* verwendet die aktuell ausgewählte Schrift, um die Abmessungen des Strings zu berechnen. Breite und Höhe werden ohne Berücksichtigung der aktuellen Clipping-Region in virtuellen Einheiten berechnet.

DC: Bezeichnet den Gerätekontext.

Str: Zeigt auf einen Textstring.

Count: Bestimmt die Anzahl von Zeichen im Textstring.

TabPositions: Bestimmt die Anzahl von Tabulatorstoppositionen in dem Array, auf das der Parameter *TabStopPositions* zeigt.

TabStopPositions: Zeigt auf ein Array von Integerwerten, das die Tabulatorstoppositionen in Pixeln enthält. Die Tabulatorstops müssen in aufsteigender Reihenfolge sortiert werden, Rückwärtstabulatoren sind nicht erlaubt.

Rückgabewert Der Rückgabewert gibt die Abmessungen des Strings an, die Höhe liegt im höherwertigen, die Breite im niederwertigen Word.

Hinweis Da einige Geräte Zeichen nicht in regulären Zellen plazieren (das heißt, sie führen *Kerning* aus), kann die Summe der Abmessungen der Zeichen in einem String eventuell von den Abmessungen des Strings verschieden sein.

Wenn die Parameter *TabPositions* und *TabStopPositions* 0 sind, werden die Tabulatoren auf acht durchschnittliche Zeichenbreiten erweitert.

Wenn *TabPositions* 1 ist, werden die Tabulatorstops mit dem Abstand getrennt, den der erste Wert in dem Array angibt, auf das *TabStopPositions* zeigt.

Wenn *TabStopPositions* auf mehr als einen einzigen Wert zeigt, wird ein Tabulatorstop für jeden Wert im Array gesetzt, bis zu der Zahl, die von *TabPositions* vorgegeben ist.

GetTempDrive

Syntax
```
function GetTempDrive(DriveLetter: Char): Char;
```

Beschreibung Diese Funktion nimmt eine Laufwerksbezeichnung oder 0 entgegen und gibt einen Kennbuchstaben zurück, der das optimale Laufwerk für eine temporäre Datei bezeichnet. Dies ist das Diskettenlaufwerk, das während Diskettenoperationen mit einer temporären Datei die beste Zugriffszeit bietet.

GetTempDrive gibt die Laufwerksbezeichnung einer Festplatte zurück, wenn das System eine hat. Ist der Parameter *DriveLetter* 0, gibt die Funktion die Laufwerksbezeichnung der aktuellen Diskette zurück. Ist *DriveLetter* ein Buchstabe, gibt die Funktion den Kennbuchstaben dieses Laufwerks oder den Kennbuchstaben eines anderen verfügbaren Laufwerks zurück.

DriveLetter: Bestimmt ein Laufwerk. 0 führt zur Rückgabe des Kennbuchstabens des aktuellen Laufwerks.

Rückgabewert Der Rückgabewert zeigt das optimale Laufwerk für temporäre Dateien an.

GetTempFileName

Syntax
```
function GetTempFileName(DriveLetter: Char;
PrefixString: PChar; Unique: Word; TempFileName:
PChar): Integer;
```

Beschreibung Diese Funktion erzeugt einen temporären Dateinamen in der folgenden Form:

```
d:\Pfad\Pr/fixuuuu.tmp
```

In dieser Syntaxzeile ist *d:* der vom Parameter *DriveLetter* vorgegebene Kennbuchstabe. *Pfad* ist der Pfadname für die temporäre Datei (entweder das Stammverzeichnis des angegebenen Laufwerks oder das in der Environment-Variablen TEMP benannte Verzeichnis); *Präfix* sind alle Buchstaben (bis zu drei) des Strings, auf den der Parameter *PrefixString* zeigt, *uuuu* ist der numerische Wert des vom Parameter *Unique* vorgegebenen Dateinamens.

DriveLetter: Bestimmt das für die temporäre Datei empfohlene Laufwerk. Wenn *DriveLetter* 0 ist, wird das Standardlaufwerk benutzt.

PrefixString: Zeigt auf einen nullterminierten String, der als Namenspräfix der temporären Datei verwendet werden soll. Dieser String muß aus Zeichen des OEM-Zeichensatzes bestehen.

Unique: Numerischer Wert des Dateinamens oder 0, wenn der Wert vom System bereitgestellt wird.

TempFileName: Zeigt auf den Puffer, welcher den temporären Dateinamen aufnehmen soll. Dieser String besteht aus Zeichen des OEM-Zeichensatzes. Der Puffer sollte mindestens 144 Bytes lang sein, um genügend Raum für den Pfadnamen zu bieten.

Rückgabewert Der Rückgabewert gibt einen eindeutigen numerischen Wert an, der im temporären Dateinamen benutzt wird. Wenn ein Wert ungleich 0 für den Parameter *Unique* angegeben war, bestimmt der Rückgabewert denselben Wert.

Hinweis Um Probleme zu vermeiden, die aus der Umwandlung von OEM-Zeichen in einen ANSI-String entstehen können, sollte die Anwendung die Funktion *_lopen* aufrufen, um die temporäre Datei zu erzeugen.

GetTempFileName benutzt die empfohlene Laufwerksbezeichnung zum Erzeugen des temporären Dateinamens, außer in den folgenden Fällen:

- Ist eine Festplatte vorhanden, benutzt *GetTempFileName* immer die Laufwerksbezeichnung der ersten Festplatte.
- Ist eine Environment-Variable *TEMP* definiert, deren Wert mit einer Laufwerksbezeichnung beginnt, wird dieser Kennbuchstabe verwendet.

Wenn das Bit *tf_ForceDrive* des Parameters *DriveLetter* gesetzt ist, gelten die oben gemachten Ausnahmen nicht. Der temporäre Dateiname wird immer im aktuellen Verzeichnis des von *DriveLetter* vorgegebenen Laufwerks erzeugt – unabhängig davon, ob eine Festplatte oder die Environment-Variable *TEMP* vorhanden sind.

Ist der Parameter *Unique* 0, versucht *GetTempFileName*, eine auf der aktuellen Systemzeit basierende eindeutige Zahl zu bilden. Existiert bereits eine Datei mit dem sich ergebenden Dateinamen, wird die Zahl um 1 erhöht und der Test auf Vorhandensein wiederholt.

Dies wird solange durchgeführt, bis ein eindeutiger Dateiname gefunden ist, *GetTempFileName* erzeugt dann eine Datei mit diesem Namen und schließt sie. Wenn *Unique* ungleich 0 ist, wird kein Versuch gemacht, die Datei zu erzeugen und zu öffnen.

GetTextAlign

Syntax

```
function GetTextAlign(DC: HDC): Word;
```

Beschreibung Diese Funktion ruft den Status der Textausrichtungsflags ab. Die Textausrichtungsflags bestimmen, wie die beiden Funktionen *TextOut* und *ExtTextOut* einen Textstring relativ zu seinem Ursprung ausrichten.

DC: Bezeichnet den Gerätekontext.

Rückgabewert Der Rückgabewert gibt den Status der Textausrichtungsflags an. Der Rückgabewert ist eine Kombination aus einem oder mehreren der folgenden Werte:

Wert	Bedeutung
ta_BaseLine	Bestimmt die Ausrichtung der x-Achse der ausgewählten Schrift an der Grundlinie des umgebenden Rechtecks.
ta_Bottom	Bestimmt die Ausrichtung der x-Achse zur Unterseite des umgebenden Rechtecks.
ta_Center	Bestimmt die Ausrichtung der y-Achse zum Mittelpunkt des umgebenden Rechtecks.
ta_Left	Bestimmt die Ausrichtung der y-Achse zur linken Seite des umgebenden Rechtecks.

Wert	Bedeutung
ta_NoUpdateCP	Bestimmt, daß die aktuelle Position nicht aktualisiert wird.
ta_Right	Bestimmt die Ausrichtung der y-Achse zur rechten Seite des umgebenden Rechtecks.
ta_Top	Bestimmt die Ausrichtung der x-Achse zur Oberseite des umgebenden Rechtecks.
ta_UpdateCP	Bestimmt, daß die aktuelle Position aktualisiert wird.

Hinweis

Textausrichtungsflags sind nicht unbedingt Einbitflags und können gleich 0 sein. Um zu überprüfen, ob ein einzelnes Flag im Rückgabewert dieser Funktion gesetzt ist, muß man eine Anwendung erstellen, die folgende Schritte ausführt:

1. Anwendung des bitweisen Operators **or** auf das Flag und seine verknüpften Flags.
   Die folgende Liste zeigt die Gruppen verknüpfter Flags:

ta_Left, *ta_Center* und *ta_Right*
ta_BaseLine, *ta_Bottom* und *ta_Top*
ta_NoUpdateCP und *ta_UpdateCP*

2. Anwendung des bitweisen Operators **and** auf das Ergebnis und den Rückgabewert.

3. Test auf Gleichheit dieses Ergebnisses und des Flags.

Das folgende Beispiel zeigt eine Methode, um zu bestimmen, welches horizontale Ausrichtungsflag gesetzt ist:

```
case (ta_Left   or ta_Right   or ta_Center)   and
     GetTextAlign(hDC)     of
  ta_Left: {......}
  ta_Right: {......}
  ta_Center: {......}
end ;
```

Querverweis *TextOut, ExtTextOut*

GetTextCharacterExtra

Syntax

```
function GetTextCharacterExtra(DC: HDC): Integer;
```

Beschreibung

Diese Funktion ruft den aktuellen Zeichenzwischenraum ab. Der Zeichenzwischenraum definiert den zusätzlichen Raum (in virtuellen Einheiten), den die Funktionen *TextOut* und *ExtTextOut* an jedes Zeichen anhängen, wenn sie eine Zeile schreiben. Das Einfügen von Leerraum dient zum Dehnen von Textzeilen.

Ist der aktuelle Abbildungsmodus nicht *mm_Text*, formt *GetTextCharacterExtra* das Ergebnis um und rundet es auf die nächste Einheit.

DC: Bezeichnet den Gerätekontext.

Rückgabewert Der Rückgabewert gibt den aktuellen Zeichenzwischenraum an.

Querverweis *TextOut, ExtTextOut*

GetTextColor

Syntax
```
function GetTextColor(DC: HDC): TColorRef;
```

Beschreibung Diese Funktion ruft die aktuelle Textfarbe ab.

Die Textfarbe definiert die Vordergrundfarbe der Zeichen, die mit einer der Funktionen *TextOut* oder *ExtTextOut* dargestellt werden.

DC: Bezeichnet den Gerätekontext.

Rückgabewert Der Rückgabewert gibt die aktuelle Textfarbe als RGB-Farbwert an.

Querverweis *TextOut, ExtTextOut*

GetTextExtent

Syntax
```
function GetTextExtent(DC: HDC; Str: PChar; Count:
Integer): LongInt;
```

Beschreibung Diese Funktion berechnet Breite und Höhe der Textzeile, auf die der Parameter *Str* zeigt.

GetTextExtent verwendet die aktuell ausgewählte Schrift, um die Abmessungen des Strings zu berechnen. Breite und Höhe werden ohne Berücksichtigung der aktuellen Clipping-Region in virtuellen Einheiten berechnet.

DC: Bezeichnet den Gerätekontext.

Str: Zeigt auf einen Textstring.

Count: Bestimmt die Anzahl von Zeichen im Textstring.

Rückgabewert Der Rückgabewert gibt die Abmessungen des Strings an. Die Höhe liegt im niederwertigen, die Breite im höherwertigen Word.

Hinweis Da einige Geräte Zeichen nicht in regulären Zellen abbilden (das heißt, sie führen *Kerning* aus), kann die Summe der Zeichenabmessungen in einem String eventuell von den Abmessungen des Strings abweichen.

GetTextFace

Syntax
```
function GetTextFace(DC: HDC; Count: Integer; Facename:
PChar): Integer;
```

Beschreibung Diese Funktion kopiert die Schriftartbezeichnung der ausgewählten Schrift in einen Puffer, auf den der Parameter *Facename* zeigt.

Die Schriftartenbezeichnung wird als nullterminierter String kopiert. Der Parameter *Count* gibt die maximale Anzahl von Zeichen an, die kopiert werden sollen. Wenn die Schriftartbezeichnung länger ist als die von *Count* vorgegebene Zahl, wird sie abgeschnitten.

DC: Bezeichnet den Gerätekontext.

Count: Bestimmt die Größe des Puffers in Byte.

Facename: Zeigt auf den Puffer, der die Schriftartbezeichnung aufnehmen soll.

Rückgabewert Der Rückgabewert gibt die Anzahl der tatsächlich in den Puffer kopierten Zeichen an. Er ist 0, wenn ein Fehler auftritt.

GetTextMetrics

Syntax
```
function GetTextMetrics(DC: HDC; var Metrics:
TTextMetric): Bool;
```

Beschreibung Diese Funktion füllt die *TTextMetric*-Struktur *Metrics* mit den Abmessungen der ausgewählten Schrift.

DC: Bezeichnet den Gerätekontext.

Metrics: TTextMetric-Datenstruktur, die die Abmessungen aufnehmen soll.

Rückgabewert Der Rückgabewert ist *True*, wenn die Funktion erfolgreich war, andernfalls ist er *False*.

GetThresholdEvent

Syntax
```
function GetThresholdEvent: PInteger;
```

Beschreibung Diese Funktion ruft ein Flag ab, das ein kürzlich eingetretenes Schwellenereignis bezeichnet.

Ein Schwellenereignis ist jeder Übergang einer Ton-Warteschlange von *n* auf *n-1*, wobei *n* der Schwellenwert in Tönen ist.

Diese Funktion hat keine Parameter.

Rückgabewert Der Rückgabewert ist ein Zeiger auf ein Schwellenereignis.

GetThresholdStatus

Syntax
```
function GetThresholdStatus: Integer;
```

Beschreibung Diese Funktion ruft den Schwellenereignisstatus für jeden Ton ab. Jedes Bit im Status stellt einen Ton dar. Wenn ein Bit gesetzt ist, liegt der Ton-Warteschlangenwert gegenwärtig unter der Schwelle. *GetThresholdStatus* löscht auch das Schwellenereignisflag.

Diese Funktion hat keine Parameter.

Rückgabewert Der Rückgabewert gibt die Statusflags des aktuellen Schwellenereignisses an.

GetTickCount

Syntax
```
function GetTickCount: LongInt;
```

Beschreibung Diese Funktion ermittelt die Anzahl von Millisekunden, die vergangen sind, seit das System gestartet wurde.

Diese Funktion hat keine Parameter.

Rückgabewert Der Rückgabewert gibt die Anzahl von Millisekunden an, die vergangen sind, seit das System gestartet wurde.

Hinweis Dieser Wert ist auf ca. 55 Millisekunden (+/-) genau.

GetTopWindow

Syntax
```
function GetTopWindow(Wnd: HWnd): HWnd;
```

Beschreibung Diese Funktion sucht nach einem Handle zum obersten untergeordneten Fenster, das zu dem übergeordneten Fenster *Wnd* gehört. Hat das Fenster keine untergeordneten Fenster, gibt die Funktion 0 zurück.

Wnd: Bezeichnet das übergeordnete Fenster.

Rückgabewert Der Rückgabewert bezeichnet ein Handle zu dem obersten untergeordneten Fenster in der verketteten Liste der untergeordneten Fenster eines übergeordneten Fensters. Wenn kein untergeordnetes Fenster existiert, ist er 0.

GetUpdateRect

Syntax
```
function GetUpdateRect(Wnd: HWnd; var Rect: TRect;
Erase: Bool): Bool;
```

Beschreibung Diese Funktion ruft die Koordinaten des kleinsten Rechtecks ab, das die Aktualisierungsregion des angegebenen Fensters vollständig umschließt.

Wenn das Fenster mit dem Stil *cs_OwnDC* erstellt wurde und der Abbildungsmodus nicht *mm_Text* ist, bestimmt *GetUpdateRect* das Rechteck in virtuellen Koordinaten, andernfalls in Client-Koordinaten. Gibt es keine Aktualisierungsregion, leert *GetUpdateRect* das Rechteck (d.h. setzt alle Koordinaten auf 0).

Der Parameter *Erase* gibt an, ob *GetUpdateRect* den Hintergrund der Aktualisierungsregion löschen soll. Wenn *Erase True* und die Aktualisierungsregion nicht leer ist, wird der Hintergrund gelöscht. Hierzu sendet *GetUpdateRect* eine *wm_EraseBkgnd*-Botschaft an das angegebene Fenster.

Wnd: Bezeichnet das Fenster, dessen Aktualisierungsregion abgerufen werden soll.

Rect: TRect-Datenstruktur, die die Client-Koordinaten des umschließenden Rechtecks aufnehmen soll.

Erase: Bestimmt, ob der Hintergrund in der Aktualisierungsregion gelöscht werden soll.

Rückgabewert Der Rückgabewert gibt den Status der Aktualisierungsregion des angegebenen Fensters an. Er ist *True*, wenn die Aktualisierungsregion nicht leer ist, andernfalls ist er *False*.

Hinweis Das Aktualisierungsrechteck, das von der Funktion *BeginPaint* abgerufen wird, ist identisch mit dem, das von der Funktion *GetUpdateRect* abgerufen wird.

BeginPaint prüft die Aktualisierungsregion automatisch auf Gültigkeit, so daß ein Aufruf von *GetUpdateRect* unmittelbar nach dem Aufruf von *BeginPaint* eine leere Aktualisierungsregion liefert.

Querverweis *wm_EraseBkgnd*

GetUpdateRgn

Syntax
```
function GetUpdateRgn(Wnd: HWnd; Rgn: HRgn; Erase:
Bool): Integer;
```

Beschreibung Diese Funktion kopiert die Aktualisierungsregion eines Fensters in die durch den Parameter *Rgn* bezeichnete Region. Die Koordinaten dieser Region werden relativ zur oberen linken Ecke des Fensters (in Client-Koordinaten) gebildet.

Wnd: Bezeichnet das Fenster, das die Aktualisierungsregion enthält.

Rgn: Bezeichnet die Aktualisierungsregion.

Erase: Bestimmt, ob der Fensterhintergrund gelöscht werden soll und ob Nicht-Client-Bereiche von untergeordneten Fenstern gezeichnet werden sollen. Wenn er *False* ist, wird nicht gezeichnet.

Rückgabewert Der Rückgabewert gibt den Typ der resultierenden Region an. Er kann einen der folgenden Werte annehmen:

Wert	Bedeutung
ComplexRegion	Die Region hat überlappende Grenzen.
Error	Es wurde keine Region erzeugt.
NullRegion	Die Region ist leer.
SimpleRegion	Die Region hat keine überlappenden Grenzen.

Hinweis *BeginPaint* prüft die Aktualisierungsregion automatisch auf Gültigkeit, so daß ein Aufruf von *GetUpdateRgn* unmittelbar nach dem Aufruf von *BeginPaint* eine leere Aktualisierungsregion liefert.

GetVersion

Syntax
```
function GetVersion: Word;
```

Beschreibung Diese Funktion ermittelt die aktuelle Versionsnummer von Windows.
Die Funktion hat keine Parameter.

Rückgabewert Der Rückgabewert gibt die Haupt- und Nebenversionsnummer von Windows an. Das höherwertige Word gibt die Nebenversionsnummer (Revision), das niederwertige die Hauptversionsnummer an.

GetViewportExt

Syntax
```
function GetViewportExt(DC: HDC): LongInt;
```

Beschreibung Diese Funktion ruft die x- und y-Abmessungen des Grafikfensters des Gerätekontexts ab.

DC: Bezeichnet den Gerätekontext.

Rückgabewert Der Rückgabewert gibt die x- und y-Abmessungen in gerätebezogenen Einheiten) an. Die y-Abmessung liegt im höherwertigen, die x-Abmessung im niederwertigen Word.

GetViewportOrg

Syntax
```
function GetViewportOrg(DC: HDC): LongInt;
```

Beschreibung Diese Funktion ruft die x- und y-Koordinaten des Ursprungs des Grafikfensters ab, das dem angegebenen Gerätekontext zugeordnet ist.

DC: Bezeichnet den Gerätekontext.

Rückgabewert Der Rückgabewert zeigt den Ursprung des Grafikfensters in gerätebezogenen Koordinaten an. Die y-Koordinate liegt im höherwertigen, die x-Koordinate im niederwertigen Word.

GetWindow

Syntax
```
function GetWindow(Wnd: HWnd; Cmd: Word): HWnd;
```

Beschreibung Diese Funktion sucht nach einem Handle zu einem Fenster aus der Liste des Fenster-Managers. Die Liste des Fenster-Managers enthält Einträge für alle Hauptfenster, ihre zugehörenden untergeordneten Fenster sowie deren untergeordnete Fenster. Der Parameter *Cmd* gibt die Beziehung zwischen dem vom Parameter *Wnd* bezeichneten Fenster und dem Fenster, dessen Handle zurückgegeben wird, an.

Wnd: Bezeichnet das ursprüngliche Fenster.

Cmd: Gibt die Beziehung zwischen dem ursprünglichen Fenster und dem zurückgegebenen an. Er kann einen der folgenden Werte annehmen:

Wert	Bedeutung
gw_Child	Bezeichnet das erste untergeordnete Fenster des Fensters.
gw_HWndFirst	Gibt das erste Nachbarfenster eines untergeordneten Fensters an. Existiert kein solches Fenster, wird das erste Hauptfenster in der Liste angezeigt.
gw_HWndLast	Gibt das letzte Nachbarfenster für ein untergeordnetes Fenster oder das letzte Hauptfenster in der Liste an.
gw_HWndNext	Gibt das Fenster an, das dem vorgegebenen Fenster in der Liste des Fenster-Managers folgt.
gw_HWndPrev	Gibt das vorhergehende Fenster in der Liste des Fenster-Managers an.
gw_Owner	Bezeichnet den Besitzer des Fensters.

Rückgabewert Der Rückgabewert bezeichnet ein Fenster. Er ist 0, wenn die Funktion das Ende der Liste des Fenster-Managers erreicht hat oder wenn der Parameter *Cmd* ungültig ist.

GetWindowDC

Syntax
```
function GetWindowDC(Wnd: HWnd): HDC;
```

Beschreibung Diese Funktion ruft den Bildschirmkontext für das gesamte Fenster ab, einschließlich Titelzeile, Menüs und Bildlaufleisten. Ein Fenster-Bildschirmkontext erlaubt es, irgendwo in

einem Fenster zu zeichnen – also auch in der Titelzeile, den Menüs und Bildlaufleisten – da der Ursprung des Kontexts die linke obere Ecke des Fensters ist, nicht die des Client-Bereichs.

GetWindowDC weist dem Bildschirmkontext immer, wenn sie den Kontext abruft, Standardattribute zu. Vorherige Attribute gehen verloren.

Wnd: Bezeichnet das Fenster, dessen Bildschimrkontext abgerufen werden soll.

Rückgabewert Der Rückgabewert bezeichnet den Bildschirmkontext für das angegebene Fenster, wenn die Funktion erfolgreich war, andernfalls ist er 0.

Hinweis Die Funktion *GetWindowDC* ist für bestimmte Zeicheneffekte innerhalb des Nicht-Client-Bereichs eines Fensters vorgesehen. Das Zeichnen in Nicht-Client-Bereichen ist jedoch nicht empfehlenswert.

Die Funktion *GetSystemMetrics* kann dazu benutzt werden, die Abmessungen verschiedener Teile des Nicht-Client-Bereichs, wie zum Beispiel der Titelzeile, des Menüs und der Bildlaufleisten, abzurufen.

Nachdem das Zeichnen beendet ist, muß die Funktion *ReleaseDC* aufgerufen werden, um den Bildschirmkontext freizugeben. Wird er nicht freigegeben, hat dies fatale Auswirkungen auf Zeichenvorgänge, die in anderen Anwendungen erforderlich sind.

Querverweis *ReleaseDC*

GetWindowExt

Syntax
```
function GetWindowExt(DC: HDC): LongInt;
```

Beschreibung Diese Funktion ruft die x- und y-Abmessungen des Fensters ab, das dem angegebenen Gerätekontext zugeordnet ist.

DC: Bezeichnet den Gerätekontext.

Rückgabewert Der Rückgabewert gibt die x- und y-Abmessungen in virtuellen Einheiten an. Die y-Abmessung liegt im höherwertigen, die x-Abmessung im niederwertigen Word.

GetWindowLong

Syntax
```
function GetWindowLong(Wnd: HWnd; Index: Integer):
LongInt;
```

Beschreibung Diese Funktion ruft Informationen über das vom Parameter *Wnd* bezeichnete Fenster ab.

Wnd: Bezeichnet das Fenster.

Index: Gibt den Byte-Offset des abzurufenden Wertes an. Es kann auch einer der folgenden Werte sein:

Wert:	Bedeutung
gwl_ExStyle	Erweiterter Fensterstil.
gwl_Style	Fensterstil.
gwl_WndProc	Zeiger auf die Fensterfunktion.

Rückgabewert Der Rückgabewert liefert Informationen über das angegebene Fenster.

Hinweis Um auf alle zusätzlichen Vier-Byte-Werte zuzugreifen, die beim Erzeugen der Fensterklassenstruktur belegt wurden, benutzt man einen positiven Byte-Offset ab dem vom Parameter *Index* bezeichneten Index. Dabei wird mit 0 für den ersten Vier-Byte-Wert im zusätzlichen Speicherraum begonnen, 4 für den nächsten Vier-Byte-Wert, und so weiter.

GetWindowOrg

Syntax
```
function GetWindowOrg(DC: HDC): LongInt;
```

Beschreibung Diese Funktion ruft die x- und y-Koordinaten des Ursprungs des Fensters ab, das dem angegebenen Gerätekontext zugeordnet ist.

DC: Bezeichnet den Gerätekontext.

Rückgabewert Der Rückgabewert gibt den Ursprung des Fensters in virtuellen Koordinaten an.

Die y-Koordinate liegt im höherwertigen, die x-Koordinate im niederwertigen Word.

GetWindowRect

Syntax
```
procedure GetWindowRect(Wnd: HWnd; var Rect: TRect);
```

Beschreibung Diese Prozedur kopiert die Abmessungen des umgrenzenden Rechtecks des angegebenen Fensters in die Struktur, auf die der Parameter *Rect* zeigt.

Die Abmessungen sind in Bildschirmkoordinaten relativ zur oberen linken Ecke des Bildschirms angegeben und schließen die Kopfzeile, die Umrandung und die Bildlaufleisten – falls diese vorhanden sind – ein.

Wnd: Bezeichnet das Fenster.

Rect: Eine *TRect*-Datenstruktur, die die Bildschirmkoordinaten der oberen linken und unteren rechten Ecke des Fensters enthält.

GetWindowsDirectory

Syntax
```
function GetWindowsDirectory(Buffer: PChar; Size: Word): Word;
```

Beschreibung Diese Funktion ermittelt den Pfadnamen des Windows-Verzeichnisses. Das Windows-Verzeichnis enthält unter anderem Windows-Anwendungen, Initialisierungsdateien und Hilfedateien.

Buffer: Zeigt auf den Puffer, der den nullterminierten String aufnehmen soll, der den Pfadnamen enthält.

Size: Bestimmt die maximale Größe des Puffers in Bytes. Dieser Wert sollte auf mindestens 144 gesetzt werden, um genügend Platz für den Pfadnamen im Puffer bereitzustellen.

Rückgabewert Der Rückgabewert ist die Länge des Strings, der in *Buffer* kopiert wurde, ohne Berücksichtigung des abschließenden Null-Zeichens.

Wenn der Rückgabewert größer als *Size* ist, entspricht er der Größe des Puffers, die erforderlich ist, um den Pfadnamen aufzunehmen. Der Rückgabewert ist 0, wenn die Funktion fehlschlägt.

Hinweis Der von dieser Funktion abgerufene Pfadname endet nicht mit einem Backslash (\), wenn das Windows-Verzeichnis nicht das Hauptverzeichnis des Laufwerks ist.

Wenn z.B. das Windows-Verzeichnis auf Laufwerk C: mit *WINDOWS* benannt wurde, so lautet der Pfadname des von der Funktion abgerufenen Windows-Verzeichnisses *C:\WINDOWS*. Wenn Windows im Stammverzeichnis von Laufwerk C: installiert ist, ist der von dieser Funktion abgerufene Pfadname *C:*.

GetWindowTask

Syntax
```
function GetWindowTask(Wnd: HWnd): THandle;
```

Beschreibung Diese Funktion sucht nach dem Handle einer Task, die dem Parameter *Wnd* zugeordnet ist.

Eine Task ist jedes Programm, das als unabhängige Einheit arbeitet. Alle Anwendungen werden als Task ausgeführt. Jede Instanz einer Anwendung ist eine Task.

Wnd: Bezeichnet das Fenster, für das ein Task-Handle abgerufen wird.

Rückgabewert Der Rückgabewert bezeichnet die Task, die einem bestimmten Fenster zugeordnet ist.

GetWindowText

Syntax
```
function GetWindowText(Wnd: HWnd; Str: PChar; MaxCount:
Integer): Integer;
```

Beschreibung Diese Funktion kopiert die Überschrift des angegebenen Fensters in den Puffer, auf den der Parameter *Str* zeigt. Bezeichnet der Parameter *Wnd* ein Steuerelement, kopiert *GetWindowText* den Text innerhalb des Steuerelements und nicht die Überschrift.

Wnd: Bezeichnet das Fenster oder das Steuerelement, dessen Überschrift bzw. Text kopiert werden soll.

Str: Zeigt auf den Puffer, der den kopierten String aufnehmen soll.

MaxCount: Bestimmt die maximale Anzahl von Zeichen, die in den Puffer kopiert werden sollen. Wenn der String länger ist als die vom Parameter *MaxCount* vorgegebene Anzahl, wird er abgeschnitten.

Rückgabewert Der Rückgabewert gibt die Länge des kopierten Strings an. Er ist 0, wenn das Fenster keine Überschrift hat oder wenn die Kopfzeile leer ist.

Hinweis Diese Funktion bewirkt, daß eine Botschaft *wm_GetText* an das Fenster oder Steuerelement gesendet wird.

GetWindowTextLength

Syntax
```
function GetWindowTextLength(Wnd: HWnd): Integer;
```

Beschreibung Diese Funktion gibt die Länge der Überschrift des angegebenen Fensters zurück.

Wenn der Parameter *Wnd* ein Steuerelement bezeichnet, gibt die Funktion die Länge des Textes innerhalb des Steuerelements zurück.

Wnd: Bezeichnet das Fenster oder Steuerelement.

Rückgabewert Der Rückgabewert gibt die Textlänge an. Er ist 0, wenn kein entsprechender Text existiert.

GetWindowWord

Syntax
```
function GetWindowWord(Wnd: HWnd; Index: Integer): Word;
```

Beschreibung Diese Funktion ruft Informationen über das von *Wnd* bezeichnete Fenster ab.

Wnd: Bezeichnet das Fenster.

Index: Bestimmt den Byte-Offset des abzurufenden Wertes. Er kann auch einen der folgenden Werte annehmen:

Wert	Bedeutung
gww_HInstance	Instanz-Handle des Moduls, das das Fenster besitzt.
gww_HWndParent	Handle des übergeordneten Fensters, wenn dieses vorhanden ist. Die Funktion *SetParent* ändert das übergeordnete Fenster eines untergeordneten Fensters. Eine Anwendung sollte nicht die Funktion *SetWindowLong* aufrufen, um das übergeordnete Fenster eines untergeordneten Fensters zu ändern.
gww_ID	Steuerelement-ID des untergeordneten Fensters.

Rückgabewert Dieser gibt Informationen über das bezeichnete Fenster an.

Hinweis Um auf zusätzliche Zwei-Byte-Werte zuzugreifen, die beim Erzeugen der Fensterklassenstruktur belegt wurden, benutzt man einen positiven Byte-Offset ab dem vom Parameter *Index* bezeichneten Index – beginnend bei 0 für den ersten Zwei-Byte-Wert im zusätzlichen Speicher, 2 für den nächsten Zwei-Byte-Wert, und so weiter.

GetWinFlags

Syntax
```
function GetWinFlags: LongInt;
```

Beschreibung Diese Funktion gibt einen 32-Bit-Wert zurück, der die Flags enthält, die die Speicherkonfiguration beschreiben, unter der Windows läuft. Diese Funktion hat keine Parameter.

Rückgabewert Der Rückgabewert enthält Flags, die die aktuelle Speicherkonfiguration angeben. Sie können folgende Werte haben:

Wert	Bedeutung
wf_80x87	System enthält einen mathematischen Intel-Coprozessor.
wf_CPU086	System-CPU ist ein 8086.
wf_CPU186	System-CPU ist ein 80186.
wf_CPU286	System-CPU ist ein 80286.
wf_CPU386	System-CPU ist ein 80386.
wf_CPU486	System-CPU ist ein 80486.
wf_Enhanced	Windows läuft im erweiterten 386-Modus. Das Flag *wf_PMode* ist immer gesetzt, wenn *wf_Enhanced* gesetzt ist.

Wert	Bedeutung
wf_LargeFrame	Windows läuft mit Large-Frame-EMS-Speicherkonfiguration.
wf_PMode	Windows läuft im Protected Mode. Dieses Flag ist immer gesetzt, wenn entweder *wf_Enhanced* oder *wf_Standard* gesetzt ist.
wf_SmallFrame	Windows läuft mit Small-Frame-EMS-Speicherkonfiguration.
wf_Standard	Windows läuft im Standardmodus. Das Flag *wf_PMode* ist immer gesetzt, wenn *wf_Standard* gesetzt ist.

Wenn weder *wf_Enhanced* noch *wf_Standard* gesetzt sind, läuft Windows im Real Mode.

GlobalAddAtom

Syntax
```
function GlobalAddAtom(Str: PChar): TAtom;
```

Beschreibung Diese Funktion fügt den String, auf den der Parameter *Str* zeigt, zur Atomtabelle hinzu und erzeugt ein neues globales Atom, das den String eindeutig bezeichnet.

Ein globales Atom ist ein Atom, das für alle Anwendungen verfügbar ist. Das Atom kann in einer nachfolgenden Funktion *GlobalGetAtomName* benutzt werden, um den String aus der Atomtabelle abzurufen.

GlobalAddAtom speichert nicht mehr als eine Kopie des gegebenen Strings in der Atomtabelle. Wenn der String schon in der Tabelle existiert, gibt die Funktion den existierenden Atomwert zurück und erhöht den Referenzzähler des Strings um 1. Der Referenzzähler des Strings ist eine Zahl, die angibt, wie oft *GlobalAddAtom* für einen einzelnen String aufgerufen wurde.

Str: Zeigt auf den String, der zur Atomtabelle hinzugefügt werden soll. Der String muß ein nullterminierter String sein.

Rückgabewert Der Rückgabewert bezeichnet das neu erzeugte Atom, wenn die Funktion erfolgreich war. Andernfalls ist er 0.

Hinweis Die von *GlobalAddAtom* zurückgegebenen Atomwerte liegen innerhalb des Bereiches von $C000 bis $FFFF.

GlobalAlloc

Syntax
```
function GlobalAlloc(Flags: Word; Bytes: LongInt):
THandle;
```

Beschreibung Diese Funktion reserviert die vom Parameter *Bytes* vorgegebene Zahl von Bytes im globalen Heap. Der Speicher kann fest oder verschiebbar sein, abhängig von dem durch den Parameter *Flags* angegebenen Speichertyp.

Flags: Gibt ein oder mehrere Flags an, die der Funktion *GlobalAlloc* mitteilen, wie der Speicher reserviert werden soll. Folgende Werte sind möglich:

Wert	Bedeutung
gmem_DDEShare	Reserviert gemeinsam nutzbaren Speicher. Wird nur für dynamischen Datenaustausch (DDE) verwendet. Es ist jedoch zu beachten, daß Windows automatisch Speicher verwirft, der mit diesem Attribut reserviert wurde, wenn die Anwendung, die ihn reserviert hat, die Ausführung beendet.
gmem_Discardable	Reserviert verwerfbaren Speicher. Kann nur mit *gmem_Moveable* benutzt werden.
gmem_Fixed	Reserviert festen Speicher.
gmem_Moveable	Reserviert verschiebbaren Speicher. Kann nicht mit *gmem_Fixed* benutzt werden.
gmem_NoCompact	Verdichtet und verwirft nicht, um den Belegungsbedarf zu befriedigen.
gmem_NoDiscard	Verwirft nicht, um den Belegungsbedarf zu befriedigen.
gmem_Not_Banked	Belegt Speicher, der nicht in Bänke aufgeteilt ist. Kann nicht mit *gmem_Notify* benutzt werden.
gmem_Notify	Ruft die Benachrichtigungsroutine auf, falls das Speicherobjekt verworfen wird.
gmem_Zeroinit	Initialisiert Speicherinhalte mit 0. Wählen Sie *gmem_Fixed* oder *gmem_Moveable* aus und kombinieren dann andere wie gewünscht mit dem bitweisen **or**-Operator.

Bytes: Bestimmt die Zahl von Bytes, die reserviert werden sollen.

Rückgabewert Der Rückgabewert bezeichnet den reservierten globalen Speicher, wenn die Funktion erfolgreich war. Andernfalls ist er 0.

Hinweis Wenn diese Funktion erfolgreich war, reserviert sie mindestens den angeforderten Speicherplatz. Der tatsächlich reservierte Speicherplatz kann größer sein und die Anwendung kann den gesamten Bereich nutzen. Um die tatsächlich reservierte Speichermenge zu bestimmen, rufen Sie die Funktion *GlobalSize* auf.

Der größte Speicherblock den eine Anwendung belegen

kann, umfaßt 1 MByte im Standardmodus und 64 MByte im erweiterten 386-Modus.

Querverweis *GlobalSize*

GlobalCompact

Syntax
```
function GlobalCompact(MinFree: LongInt): LongInt;
```

Beschreibung Diese Funktion erzeugt die Anzahl freier Bytes im globalen Speicher, die der Parameter *MinFree* bezeichnet, indem sie den globalen Heap des Systems verdichtet und gegebenenfalls Teile des Heaps verwirft.

Die Funktion verdichtet immer zuerst den Speicher, bevor sie nach freiem Speicherbereich sucht. Sie überprüft dann den globalen Speicher auf einen Bereich aufeinanderfolgender freier Bytes, in der Größe, die der Parameter *MinFree* vorgibt. Wenn kein solcher Bereich vorhanden ist, verwirft die Funktion *GlobalCompact* ungesperrte verwerfbare Blöcke, bis der erforderliche Platz erzeugt ist.

MinFree: Bestimmt die gewünschte Anzahl freier Bytes.

Rückgabewert Der Rückgabewert gibt die Anzahl von Bytes im größten Block freien Speichers an.

Hinweis Wenn *MinFree* 0 ist, gibt der Rückgabewert die Zahl von Bytes im größten freien Segment an, das Windows erzeugen kann, wenn es alle verwerfbaren Segmente löscht.

Wenn eine Anwendung den Rückgabewert als Parameter *Bytes* für die Funktion *GlobalAlloc* benutzt, dann sollten die Flags *gmem_Nocompact* und *gmem_Nodiscard* nicht benutzt werden.

GlobalDeleteAtom

Syntax
```
function GlobalDeleteAtom(AnAtom: TAtom): TAtom;
```

Beschreibung Diese Funktion verringert den Referenzzähler eines globalen Atoms um 1. Wenn der Referenzzähler des Atoms 0 wird, löscht diese Funktion den zugeordneten String aus der Atomtabelle. Ein globales Atom ist ein Atom, das für alle Windows-Anwendungen verfügbar ist.

Der Referenzzähler eines Atoms gibt an, wie oft das Atom zur Atomtabelle hinzugefügt wurde. Die Funktion *GlobalAddAtom* erhöht den Zähler bei jedem Aufruf; die Funktion *GlobalDeleteAtom* löscht den String nur, wenn der Referenzzähler des Atoms 0 ist.

AnAtom: Bezeichnet das Atom und den String, der gelöscht werden sollen.

Rückgabewert Der Rückgabewert gibt das Ergebnis der Funktion an. Er ist 0, wenn die Funktion erfolgreich war. Er ist gleich *AnAtom*, wenn die Funktion fehlschlägt und das Atom nicht gelöscht wurde.

GlobalDiscard

Syntax
```
function GlobalDiscard(Mem: THandle): THandle;
```

Beschreibung Diese Funktion verwirft einen globalen Speicherblock, der vom Parameter *Mem* bezeichnet wird. Der Sperrenzähler muß 0 sein.

Der globale Speicherblock wird aus dem Speicher gelöscht, sein Handle bleibt aber gültig. Eine Anwendung kann nachfolgend das Handle an die Funktion *GlobalReAlloc* übergeben, um einen anderen globalen Speicherblock, der vom gleichen Handle bezeichnet wird, zu belegen.

Mem: Bezeichnet den globalen Speicherblock, der verworfen werden soll.

Rückgabewert Wenn die Funktion erfolgreich war, bezeichnet der Rückgabewert den verworfenen Block, andernfalls ist er 0.

Hinweis Die Funktion verwirft nur globale Objekte, die von einer Anwendung mit gesetzten Flags *gmem_Discardable* und *gmem_Moveable* zugewiesen wurden. Sie kann nicht ausgeführt werden, wenn eine Anwendung versucht, ein festes oder gesperrtes Objekt zu verwerfen.

GlobalDosAlloc

Syntax
```
function GlobalDosAlloc(Bytes: LongInt): LongInt;
```

Beschreibung Diese Funktion belegt globalen Speicher, auf den von DOS im Real Mode zugegriffen werden kann. Es wird garantiert, daß der Speicher im ersten MByte des linearen Adressraums liegt.

Bytes: Gibt die Anzahl der zu reservierenden Bytes an.

Rückgabewert Der Rückgabewert enthält einen Paragraphsegmentwert in seinem höherwertigen und einen Selektor in seinem niederwertigen Word. Eine Anwendung kann den Paragraphsegmentwert benutzen, um im Real Mode und den Selektor, um im Protected Mode auf den Speicher zuzugreifen. Läuft Windows im Real Mode, so sind das höherwertige und das niederwertige Word gleich. Wenn Windows keinen Speicherblock mit der geforderten Größe reservieren kann, so wird 0 zurückgeliefert.

Hinweis Eine Anwendung sollte diese Funktion nur dann benutzen, wenn es unbedingt notwendig ist.

GlobalDosFree

Syntax
```
function GlobalDosFree(Selector: Word): Word;
```

Beschreibung Diese Funktion gibt einen globalen Speicherblock frei, der vorher durch einen Aufruf der Funktion *GlobalDosAlloc* reserviert wurde.

Selector: Gibt den Speicher an, der freigegeben werden soll.

Rückgabewert Bei erfolgreicher Ausführung wird 0 zurückgeliefert, ansonsten ist der Rückgabewert gleich *Selector*.

GlobalFindAtom

Syntax
```
function GlobalFindAtom(Str: PChar): TAtom;
```

Beschreibung Diese Funktion sucht in der Atomtabelle nach dem String, auf den der Parameter *Str* zeigt, und ruft das globale Atom ab, das diesem String zugeordnet ist.

Ein globales Atom ist ein Atom, das allen Windows-Anwendungen verfügbar ist.

Str: Zeigt auf den String, nach dem gesucht werden soll. Der String muß ein nullterminierter String sein.

Rückgabewert Der Rückgabewert bezeichnet das globale Atom, das dem gegebenen String zugeordnet ist. Er ist 0, wenn der String nicht in der Tabelle ist.

GlobalFix

Syntax
```
procedure GlobalFix(Mem: THandle);
```

Beschreibung Diese Prozedur verhindert, daß der vom Parameter *Mem* bezeichnete globale Speicherblock im linearen Speicher verschoben wird.

Der Block wird im linearen Speicher an seiner aktuellen Adresse gesperrt und sein Sperrenzähler wird um 1 erhöht. Gesperrter Speicher wird nicht verschoben oder verworfen, außer in dem Fall, daß der Speicherblock von der Funktion *GlobalReAlloc* neu zugewiesen wird. Der Block bleibt solange gesperrt im Speicher, bis sein Sperrenzähler auf 0 zurückgesetzt wurde.

Jedesmal, wenn eine Anwendung *GlobalFix* für ein Speicherobjekt aufgerufen hat, muß sie entsprechend *GlobalUnfix* für das Objekt aufrufen. Die Funktion *GlobalUnfix* verringert den Sperrenzähler für das Objekt.

Andere Funktionen können den Sperrenzähler eines Speicherobjektes ebenfalls beeinflussen. Die Beschreibung von

GlobalFlags enthält eine Liste der Funktionen, die den Sperrenzähler beeinflussen.

Mem: Bezeichnet den globalen Speicherblock.

Hinweis

Der Aufruf dieser Prozedur überschneidet sich mit der Windows-Speicherverwaltung und führt zu einer Fragmentierung des linearen Adressraums. Nur sehr wenige Anwendungen benötigen Speicher im linearen Adreßraum.

GlobalFlags

Syntax

```
function GlobalFlags(Mem: THandle): Word;
```

Beschreibung

Diese Funktion gibt Informationen über den globalen Speicherblock zurück, den der Parameter *Mem* bezeichnet.

Mem: Bezeichnet den globalen Speicherblock.

Rückgabewert

Der Rückgabewert gibt ein Speicherbelegungsflag im höheren Byte an.

Das Flag kann einen der folgenden Werte annehmen:

Wert	Bedeutung
gmem_DDEShare	Der Block kann gemeinsam genutzt werden. Dies wird nur für dynamischen Datenaustausch (DDE) benötigt.
gmem_Discardable	Der Block kann verworfen werden.
gmem_Discarded	Der Block wurde verworfen.
gmem_Not_Banked	Der Block kann nicht in eine Bank aufgenommen werden.

Das untere Byte des Rückgabewerts enthält den Sperrenzähler des Blocks. Verwenden Sie die Maske *gmem_LockCount*, um den Wert des Sperrenzählers aus dem Rückgabewert zu ermitteln.

Hinweis

Um zu testen, ob ein Objekt verworfen werden kann oder nicht, verknüpft man den Rückgabewert von *GlobalFlags* mit gmem_Discardable über den **and**-Operator.

Die folgenden Funktionen und Prozeduren können den Sperrenzähler eines globalen Speicherblock beeinflussen:

Erhöht den Sperrenzähler	Verringert den Sperrenzähler
GlobalFix	*GlobalUnfix*
GlobalLock	*GlobalUnlock*
GlobalWire	*GlobalUnWire*
LockSegment	*UnlockSegment*

GlobalFree

Syntax
```
function GlobalFree(Mem: THandle): THandle;
```

Beschreibung Diese Funktion gibt den globalen Speicherblock frei, der vom Parameter *Mem* bezeichnet wird und macht das Handle des Speicherblocks ungültig.

Mem: Bezeichnet den globalen Speicherblock, der freigegeben werden soll.

Rückgabewert Bei erfolgreicher Ausführung wird 0 zurückgeliefert, andernfalls ist der Rückgabewert gleich *Mem*.

Hinweis *GlobalFree* darf nicht benutzt werden, um einen gesperrten Speicherblock freizugeben, also einen Speicherblock, dessen Sperrenzähler größer 0 ist. Die Beschreibung der Funktion *GlobalFlags* enthält eine Liste der Funktionen, die den Sperrenzähler beeinflussen.

GlobalGetAtomName

Syntax
```
function GlobalGetAtomName(AnAtom: TAtom; Buffer:
PChar; Size: Integer): Word;
```

Beschreibung Diese Funktion ruft eine Kopie des Strings ab, der dem Parameter *AnAtom* zugeordnet ist, und legt sie in dem Puffer ab, auf den der Parameter *Buffer* zeigt. Der Parameter *Size* bestimmt die maximale Größe des Puffers.

Ein globales Atom ist ein Atom, das allen Windows-Anwendungen verfügbar ist.

AnAtom: Bezeichnet den String, der abgerufen werden soll.

Buffer: Zeigt auf den Puffer, der den String empfangen soll.

Size: Bestimmt die maximale Größe des Puffers in Bytes.

Rückgabewert Der Rückgabewert ist die tatsächliche Anzahl der in den Puffer kopierten Bytes. Er ist 0, wenn das angegebene Atom ungültig ist.

GlobalHandle

Syntax
```
function GlobalHandle(Mem: Word): LongInt;
```

Beschreibung Diese Funktion ruft das Handle des globalen Speicherobjekts ab, dessen Segmentadresse oder Selektor vom Parameter *Mem* bezeichnet wird.

Mem: Bezeichnet die Segmentadresse oder den Selektor eines globalen Speicherobjekts.

Rückgabewert Das niederwertige Word des Rückgabewerts enthält das Handle des globalen Speicherobjekts, das höherwertige Word die Segmentadresse oder den Selektor des Speicherobjekts. Wenn für das Speicherobjekt kein Handle existiert, wird 0 zurückgeliefert.

GlobalLock

Syntax
```
function GlobalLock(Mem: THandle): Pointer;
```

Beschreibung Diese Funktion ruft einen Zeiger auf den globalen Speicherblock ab, den der Parameter *Mem* bezeichnet.

Ausgenommen bei nichtverwerfbaren Objekten im Protected Mode (Standardmodus oder erweiterter 386-Modus) wird der Block im Speicher an der angegebenen Adresse gesperrt und sein Sperrenzähler um 1 erhöht. Gesperrter Speicher kann nicht verschoben oder verworfen werden, ausgenommen die Neubelegung des Speicherblocks durch die Funktion *GlobalReAlloc*. Im Protected Mode erhöht *GlobalLock* den Sperrenzähler nur bei verwerfbaren Objekten und automatischen Datensegmenten.

Jedes Mal, wenn eine Anwendung *GlobalLock* für ein Objekt aufgerufen hat, muß sie hierfür auch *GlobalUnlock* aufrufen. Die Funktion *GlobalUnlock* verringert den Sperrenzähler für das Objekt, wenn *GlobalLock* den Sperrenzähler erhöht hat.

Andere Funktionen können den Sperrenzähler eines Objekts ebenfalls beeinflussen. Die Beschreibung der Funktion *GlobalFlags* enthält eine Liste dieser Funktionen.

Mem: Bezeichnet den globalen Speicherblock, der gesperrt werden soll.

Rückgabewert Bei erfolgreicher Ausführung zeigt der Rückgabewert auf das erste Speicherbyte im globalen Block. Wenn das Objekt verworfen wurde oder ein Fehler auftritt, wird 0 zurückgeliefert.

Hinweis Verworfene Objekte haben den Sperrenzählerwert 0.

GlobalLRUNewest

Syntax
```
function GlobalLRUNewest(Mem: THandle): THandle;
```

Beschreibung Diese Funktion verschiebt das globale Speicherobjekt, das von *Mem* bezeichnet wird, an die neueste, zuletzt benutzte (least-recently-used, LRU) Position im Speicher. Dies reduziert die Wahrscheinlichkeit, daß das Objekt bald verworfen wird, verhindert aber das Verwerfen nicht.

Mem: Bezeichnet das globale Speicherobjekt, das verschoben werden soll.

Rückgabewert Der Rückgabewert ist 0, wenn der Parameter *Mem* kein gültiges Handle bezeichnet.

Hinweis Diese Funktion ist nur nützlich, wenn das bezeichnete Speicherobjekt verwerfbar ist.

GlobalLRUOldest

Syntax
```
function GlobalLRUOldest(Mem: THandle): THandle;
```

Beschreibung Diese Funktion verschiebt das globale Speicherobjekt, das von *Mem* bezeichnet wird, an die älteste zuletzt benutzte (least-recently-used, LRU) Position im Speicher und macht es damit zum nächsten Anwärter, wenn Speicherobjekte verworfen werden.

Mem: Bezeichnet das globale Speicherobjekt, das verschoben werden soll.

Rückgabewert Der Rückgabewert ist 0, wenn der Parameter *Mem* kein gültiges Handle bezeichnet.

Hinweis Diese Funktion ist nur nützlich, wenn das bezeichnete Speicherobjekt verwerfbar ist.

GlobalNotify

Syntax
```
procedure GlobalNotify(NotifyProc: TFarProc);
```

Beschreibung Diese Prozedur installiert eine Benachrichtigungsprozedur für die aktuelle Task. Windows ruft die Benachrichtigungsprozedur immer dann auf, wenn ein globaler Speicherblock, der mit dem Flag *gmem_Notify* reserviert wurde, verworfen werden soll.

NotifyProc: Ist die Adresse der Prozedurinstanz der aktuellen Benachrichtigungsprozedur der Task.

Hinweis Eine Anwendung darf *GlobalNotify* nicht öfter als einmal pro Instanz aufrufen.

Windows ruft die Benachrichtigungsprozedur nicht auf, wenn es Speicher verwirft, der zu einer DLL (dynamischen Linkbibliothek) gehört.

Wurde das Objekt verworfen, muß die Anwendung das Flag *gmem_Notify* benutzen, wenn sie das Objekt durch Aufruf der Funktion *GlobalRealloc* neu erstellt. Andernfalls wird die Anwendung nicht benachrichtigt, wenn das Objekt wieder verworfen wird.

Wenn die Benachrichtigungsprozedur einen Wert ungleich 0 zurückgibt, verwirft Windows den globalen Speicherblock. Wenn sie 0 zurückgibt, wird der Block nicht verworfen.

Die Callback-Funktion muß in einem festen Code-Segment der DLL liegen.

Callback-Funktion

```
function NotifyProc(hMem: THandle): Bool; export;
```

NotifyProc: Ein Platzhalter für den von der Anwendung übergebenen Funktionsnamen.

hMem: Bezeichnet den globalen Speicherblock, der verworfen wurde.

Rückgabewert Die Funktion gibt einen Wert ungleich 0 zurück, wenn Windows den Speicherblock verwerfen soll und 0, wenn dieser nicht verworfen werden soll.

Hinweis Die Callback-Funktion wird nicht unbedingt im Kontext der Anwendung aufgerufen, der die Routine besitzt. Aus diesem Grund sollte die Callback-Funktion nicht vom Stack-Segment der Anwendung ausgehen. Die Callback-Funktion sollte keine Routine aufrufen, die Speicher verschieben könnte.

GlobalPageLock

Syntax

```
function GlobalPageLock(Selector: THandle): Word;
```

Beschreibung Diese Funktion erhöht den Seitensperrenzähler des Speichers, der dem angegebenen globalen Selektor zugeordnet ist. Solange sein Seitensperrenzähler ungleich 0 ist, bleiben die Daten, auf die sich der Selektor bezieht, im Speicher an der jeweiligen physikalischen Adresse seitenweise gespeichert.

Die Funktion *GlobalPageLock* erhöht den Seitensperrenzähler für den Speicherblock, die Funktion *GlobalPageUnlock* verringert ihn. Seitensperroperationen können verschachtelt sein, aber jede Seitensperrung muß durch eine zugehörige Aufhebung der Sperre ausgeglichen werden.

Selector: Bestimmt den Selektor des Speichers, der seitenweise gesperrt werden soll.

Rückgabewert Der Rückgabewert gibt den Wert des Seitensperrenzählers an, nachdem die Funktion ihn erhöht hat. Im Fall eines Fehlers wird 0 zurückgeliefert.

Hinweis Eine Anwendung sollte diese Funktion nur verwenden, wenn dies unbedingt notwendig ist.

Die Benutzung dieser Funktion verletzt alle gebräuchlichen Windows-Programmierpraktiken. Sie ist zum Gebrauch für dynamisch reservierte Daten vorgesehen, auf die von einem Interrupt zugegriffen werden muß. Aus diesem Grund darf sie auch nur von einer DLL (dynamischen Linkbibliothek) aufgerufen werden.

Querverweis *GlobalPageUnlock*

GlobalPageUnlock

Syntax
```
function GlobalPageUnlock(Selector: THandle): Word;
```

Beschreibung Diese Funktion verringert den Seitensperrenzähler für den vom Parameter *Selector* bezeichneten Speicherblock, und sie ermöglicht, daß der Speicherblock verschoben und auf Platte ausgelagert wird, wenn der Seitensperrenzähler 0 erreicht. *GlobalPageLock* erhöht den Seitensperrenzähler für den Speicherblock, *GlobalPageUnlock* verringert ihn.

Seitensperroperationen können verschachtelt sein, jeder Seitensperrung muß jedoch durch eine entsprechende Aufhebung der Sperre ausgeglichen werden. Diese Funktion kann nur von DLLs aufgerufen werden.

Selector: Gibt den Selektor des Speichers an, dessen Seitensperrenzähler verringert werden soll.

Rückgabewert Gibt den Seitensperrenzähler an, nachdem die Funktion ihn verringert hat. Im Fall eines Fehlers wird 0 zurückgeliefert.

GlobalReAlloc

Syntax
```
function GlobalReAlloc(Mem: THandle; Bytes: LongInt;
Flags: Word): THandle;
```

Beschreibung Diese Funktion reserviert den vom Parameter *Mem* bezeichneten globalen Speicherblock, indem sie seine Größe auf die vom Parameter *Bytes* vorgegebene Zahl von Bytes verringert oder erhöht.

Mem: Bezeichnet den globalen Speicherblock, der neu reserviert werden soll.

Bytes: Bezeichnet die neue Größe des Speicherblocks.

Flags: Bestimmt, wie der globale Block neu reserviert werden soll. Wenn die vorhandenen Speicherflags geändert werden können, benutzt man entweder einen oder mehrere der folgenden Flags (wenn mehrere Flags angegeben sind, verknüpft man sie mit dem bitweisen **or**-Operator):

Wert	Bedeutung
gmem_Discardable	Speicher kann verworfen werden. Nur mit *gmem_Modify* zu benutzen.
gmem_Modify	Speicherflags werden verändert. Der Parameter *Bytes* wird ignoriert. Nur zu benutzen, wenn eine Anwendung vorhandene Speicherflags verändern und den Speicherblock nicht mit einer neuen Größe reservieren soll.

Wert	Bedeutung
gmem_Moveable	Speicher ist verschiebbar. Wenn *Bytes* 0 ist, bewirkt dieses Flag, daß ein Objekt, das vorher als verschiebbar und verwerfbar reserviert wurde, verworfen wird, wenn der Sperrenzähler des Blocks 0 ist. Wenn der Block nicht verschiebbar und verwerfbar ist, kann *GlobalReAlloc* nicht ausgeführt werden. Wenn *Bytes* ungleich 0 und der von *Mem* angegebene Block fixiert ist, erlaubt dieses Flag, daß der neu reservierte Block an jede feste Stelle verschoben werden kann. Wenn ein verschiebbares Objekt gesperrt ist, erlaubt dieses Flag, daß das Objekt verschoben wird. Dies ist sogar dann der Fall, wenn das Objekt durch einen unmittelbar vorangegangenen Aufruf von *GlobalLock* gesperrt wurde. Beachten Sie, daß das Handle, das von *GlobalRealloc* zurückgegeben wird, sich in diesem Fall von dem Handle unterscheidet, das an die Funktion übergeben wurde. Dieses Flag sollte mit *gmem_Modify* benutzt werden, um einen festen Speicherblock verschiebbar zu machen.
gmem_NoCompact	Speicher wird nicht verdichtet oder verworfen, um den Belegungsbedarf zu befriedigen. Dieses Flag wird ignoriert, wenn das Flag *gmem_Modify* gesetzt ist.
gmem_NoDiscard	Objekte werden nicht verworfen, um den Belegungsbedarf zu befriedigen. Dieses Flag wird ignoriert, wenn das Flag *gmem_Modify* gesetzt ist.
gmem_Zeroinit	Wird der Block vergrößert, werden die zusätzlichen Speicherinhalte zu 0 initialisiert. Dieses Flag wird ignoriert, wenn das Flag *gmem_Modify* gesetzt ist.

Rückgabewert

Bei erfolgreicher Ausführung bezeichnet der Rückgabewert den neu reservierten globalen Speicher.

Wenn der Block nicht neu belegt werden konnte, wird 0 zurückgeliefert.

Der Rückgabewert nach erfolgreicher Ausführung ist immer identisch mit dem Parameter *Mem*, wenn nicht eine der folgenden Bedingungen erfüllt ist:

- Das Flag *gmem_Moveable* wird benutzt, um die Verschiebung eines festen Blocks an eine neue feste Position zu ermöglichen.

- Windows läuft im Standardmodus und die Neuzuweisung des Objektes überschreitet ein Vielfaches von 65519 Bytes (17 Bytes weniger als 64 KByte).
- Windows läuft im erweiterten 386-Modus und die Neuzuweisung des Objektes überschreitet ein Vielfaches von 64 KByte.

GlobalSize

Syntax
```
function GlobalSize(Mem: THandle): LongInt;
```

Beschreibung Diese Funktion ruft die aktuelle Größe des globalen Speicherblocks, den der Parameter *Mem* bezeichnet, in Bytes ab.

Mem: Bezeichnet den globalen Speicherblock.

Rückgabewert Der Rückgabewert gibt die tatsächliche Größe des angegebenen Speicherblocks (in Bytes) an. Er ist 0, wenn das gegebene Handle nicht gültig ist oder wenn das Objekt verworfen wurde.

Hinweis Die tatsächliche Größe eines Speicherblocks kann größer sein als die angeforderte Größe.

Eine Anwendung sollte die Funktion *GlobalFlags* vor dem Aufruf von *GlobalSize* aufrufen, um zu überprüfen, ob der angegebene Speicherblock nicht verworfen wurde. Ist der Speicherblock bereits verworfen, wäre der Rückgabewert von *GlobalSize* bedeutungslos.

GlobalUnfix

Syntax
```
function GlobalUnfix(Mem: THandle): Bool;
```

Beschreibung Diese Funktion hebt die Sperre des durch den Parameter *Mem* bezeichneten globalen Speicherblocks auf.

GlobalUnfix verringert den Sperrenzähler des Blocks um 1. Wenn der Sperrenzähler bis auf 0 verringert wurde, ist die Sperre des Blocks vollständig aufgehoben und der Block kann verschoben oder verworfen werden.

Andere Funktionen können den Sperrenzähler eines Speicherobjekts ebenfalls beeinflussen. Bei der Beschreibung der Funktion *GlobalFlags* finden Sie eine Liste dieser Funktionen.

Wenn eine Anwendung *GlobalFix* für ein Objekt aufruft, muß sie für dieses auch *GlobalUnfix* aufrufen.

Mem: Bezeichnet den globalen Speicherblock, dessen Sperre aufgehoben werden soll.

Rückgabewert Der Rückgabewert ist *False*, wenn der Sperrenzähler des Blocks bis auf 0 verringert wurde, andernfalls ist er *True*.

GlobalUnlock

Syntax
```
function GlobalUnlock(Mem: THandle): Bool;
```

Beschreibung Diese Funktion hebt die Sperre des durch den Parameter *Mem* bezeichneten globalen Speicherblocks auf.

Im Real Mode (oder wenn der Block verwerfbar ist) verringert *GlobalUnlock* den Sperrenzähler des Blocks um 1. Im Protected Mode verringert *GlobalUnlock* den Sperrenzähler nur bei verwerfbaren Objekten und automatischen Datensegmenten. Wenn der Sperrenzähler bis auf 0 verringert wurde, ist die Sperre des Blocks gänzlich aufgehoben und der Block kann verschoben oder verworfen werden. Die Beschreibung der Funktion *GlobalFlags* enthält eine Liste der Funktionen, die den Sperrenzähler beeinflussen.

Ruft eine Anwendung für ein Objekt *GlobalLock* auf, so muß sie für dieses auch *GlobalUnlock* aufrufen.

Mem: Bezeichnet den globalen Speicherblock, dessen Sperre aufgehoben werden soll.

Rückgabewert Der Rückgabewert gibt das Ergebnis der Funktion an. Er ist *False*, wenn der Sperrenzähler des Blocks bis auf 0 verringert wurde, andernfalls *True*. Eine Anwendung sollte sich nicht auf den Rückgabewert verlassen, um zu bestimmen, wie oft sie *GlobalUnlock* für einen Speicherblock nacheinander aufrufen muß.

GlobalUnWire

Syntax
```
function GlobalUnWire(Mem: THandle): Bool;
```

Beschreibung Diese Funktion hebt die Sperre eines Speichersegments auf, das von der Funktion *GlobalWire* gesperrt wurde, und verringert den Sperrenzähler um 1. Wenn der Sperrenzähler bis auf 0 verringert wurde, ist die Sperre des Segments vollständig aufgehoben und das Segment kann verschoben oder verworfen werden.

Es gibt auch andere Funktionen, die den Sperrenzähler eines Speicherobjekts beeinflussen können. Die Beschreibung der Funktion *GlobalFlags* enthält eine Liste dieser Funktionen.

Ruft eine Anwendung für ein Objekt *GlobalWire* auf, so muß sie für dieses auch *GlobalUnwire* aufrufen.

Mem: Bezeichnet das Segment, dessen Sperre aufgehoben werden soll.

Rückgabewert Wenn die Sperre des Speichersegments aufgehoben – also der Sperrenzähler auf 0 verringert – wurde, wird *True* zurückgeliefert, andernfalls *False*.

GlobalWire

Syntax
```
function GlobalWire(Mem: THandle): Pointer;
```

Beschreibung Diese Funktion verschiebt ein Segment in den unteren Speicherbereich und sperrt es. Sie ist sehr nützlich, wenn eine Anwendung ein Segment über längere Zeit hinweg sperren muß. Das Sperren eines Segments aus dem mittleren Speicherbereich über längere Zeit verursacht Speicherverwaltungsprobleme, da dies die Größe des größten zusammenhängenden verfügbaren Speicherblocks reduziert. *GlobalWire* verschiebt ein Segment an die unterste mögliche Adresse im Speicher und sperrt es, wodurch sie den Speicherbereich, den Windows am häufigsten benutzt, freigibt.

Ruft eine Anwendung für ein Objekt *GlobalWire* auf, so muß sie für dieses auch *GlobalUnwire* aufrufen. *GlobalUnWire* verringert den Sperrenzähler für das Objekt. Andere Funktionen können den Sperrenzähler eines Speicherobjektes ebenfalls beeinflussen. Die Beschreibung der Funktion *GlobalFlags* enthält eine Liste dieser Funktionen.

Eine Anwendung darf nicht die Funktion *GlobalUnlock* aufrufen, um die Sperre für das Objekt aufzuheben.

Mem: Segment, das verschoben und gesperrt werden soll.

Rückgabewert Der Rückgabewert zeigt auf die neue Segmentposition. Er ist 0, wenn die Funktion fehlgeschlagen ist.

Querverweis *GlobalUnWire*

GrayString

Syntax
```
function GrayString(DC: HDC; Brush: HBrush; OutputFunc:
TFarProc; Data: LongInt; Count, X, Y, Width, Height:
Integer): Bool;
```

Beschreibung Diese Funktion zeichnet an der angegebenen Stelle grauen, nicht aktivierbaren Text. Die Funktion *GrayString* zeichnet den Text, indem sie ihn in ein Speicher-Bitmap schreibt, das Bitmap grau färbt und es dann auf den Bildschirm kopiert. Die Funktion färbt den Text ungeachtet des ausgewählten Pinsels und Hintergrunds ein. *GrayString* benutzt die aktuell ausgewählte Schrift für den aktuellen Gerätekontext, den der Parameter *DC* vorgibt.

Wenn der Parameter *OutputFunc* 0 ist, benutzt die grafische Geräteschnittstelle (GDI) die Funktion *TextOut*. Dabei wird davon ausgegangen, daß der Parameter *Data* ein Zeiger auf den Wert ist, der ausgegeben werden soll. Wenn die Zeichen, die ausgegeben werden sollen, von *TextOut* nicht bearbeitet werden können (weil zum Beispiel der String als Bitmap gespei-

chert ist), muß die Anwendung eine eigene Ausgabefunktion zur Verfügung stellen.

DC: Bezeichnet den Gerätekontext.

Brush: Bezeichnet den Pinsel, der für das Graufärben verwendet werden soll.

OutputFunc: Ist die Adresse der Prozedurinstanz der von der Anwendung übergebenen Funktion, die den String zeichnen soll. Wenn die Funktion *TextOut* benutzt wird, um den String zu zeichnen, ist *OutputFunc* ein **nil**-Zeiger. Details finden Sie im folgenden Hinweis.

Data: Bezeichnet einen Zeiger auf Daten, die an die Ausgabefunktion übergeben werden sollen. Wenn der Parameter *OutputFunc* 0 ist, muß *Data* ein Zeiger auf den String sein, der ausgegeben werden soll.

Count: Bestimmt die Anzahl der auszugebenden Zeichen. Wenn der Parameter *Count* 0 ist, berechnet *GrayString* die Länge des Strings (vorausgesetzt, daß *Data* ein Zeiger auf den String ist). Ist *Count* -1 und gibt die Funktion, auf die *OutputFunc* zeigt, 0 zurück, wird das gezeigte Bild nicht grau dargestellt.

X: Bestimmt die x-Koordinate der virtuellen Startposition des Rechtecks, das den String einschließt.

Y: Bestimmt die y-Koordinate der Startposition des Rechtecks, das den String einschließt.

Width: Bestimmt die Breite des Rechtecks in virtuellen Einheiten, das den String einschließt. Wenn der Parameter *Width* 0 ist, berechnet *GrayString* die Breite des Bereichs, vorausgesetzt, daß *Data* ein Zeiger auf den String ist.

Height: Bestimmt die Höhe des Rechtecks (in virtuellen Einheiten), das den String einschließt. Wenn der Parameter *Height* 0 ist, berechnet *GrayString* die Höhe des Bereichs, vorausgesetzt, daß *Data* ein Zeiger auf den String ist.

Rückgabewert

Der Rückgabewert gibt das Ergebnis der Funktion an. Er ist *True*, wenn der String gezeichnet wurde. *False* bedeutet, daß entweder die Funktion *TextOut* oder die von der Anwendung definierte Funktion 0 zurückgegeben hat oder daß nicht genügend Speicherplatz vorhanden war, um ein Speicherbitmap zur Darstellung des nichtaktivierbaren Zustands zu erstellen.

Hinweis

Eine Anwendung kann graugefärbte Strings auf Geräten zeichnen, die eine flächige graue Farbe unterstützen, ohne dafür die Funktion *GrayString* aufzurufen. Die Systemfarbe *color_GrayText* ist die flächige graue Systemfarbe, die benutzt wird, um nichtaktivierbaren Text zu zeichnen.

Die Anwendung kann die Funktion *GetSysColor* aufrufen, um den Farbwert für *color_GrayText* abzurufen. Wenn die Farbe von 0 (Schwarz) abweicht, kann die Anwendung die Funktion *SetTextColor* aufrufen, um die Textfarbe auf diesen Farbwert zu setzen und dann den String direkt zu zeichnen. Wenn die abgerufene Farbe Schwarz ist, muß die Anwendung *GrayString* aufrufen, um den Text als nicht aktivierbar darzustellen.

Callback-Funktion

```
function OutputFunc(DC: HDC; lpData: LongInt; nCount:
Integer): Bool; export
```

OutputFunc: Ein Platzhalter für den von der Anwendung übergebenen Funktionsnamen.

hDC: Bezeichnet einen Speichergerätekontext mit einem Bitmap mit mindestens der Breite und der Höhe, die von den Parametern *Width* beziehungsweise *Height* vorgegeben ist.

lpData: zeigt auf den String, der gezeichnet werden soll.

nCount: Gibt die Anzahl von Zeichen an, die ausgegeben werden sollen.

Rückgabewert Bei erfolgreicher Ausführung der Funktion wird ein Wert ungleich 0 zurückgeliefert, andernfalls 0.

Hinweis Diese Ausgabefunktion (*OutputFunc*) muß ein Bild relativ zu den Koordinaten (0,0) statt zu (X,Y) zeichnen. Die als Parameter *lpOutputFunc* übergebene Adresse muß mit der Funktion *MakeProcInstance* erzeugt worden sein.

Der Abbildungsmodus *mm_Text* muß ausgewählt sein, bevor diese Funktion benutzt wird.

HiByte

Syntax

```
function HiByte(A: Word): Byte;
```

Beschreibung Diese Inline-Funktion liefert das höherwertige Byte des *Word*-Parameters *A*.

A: Gibt den Wert an, der konvertiert werden soll.

Rückgabewert Der Rückgabewert ist das höherwertige Byte des vorgegebenen *Word*.

Querverweis *GetSysColor, SetTextColor, color_Graytext, mm_Text*

HideCaret

Syntax

```
procedure HideCaret(Wnd: HWnd);
```

Beschreibung Diese Prozedur blendet das Caret aus, indem sie es vom Bildschirm löscht. Obwohl das Caret nicht mehr sichtbar ist, kann

es mit der Funktion *ShowCaret* wieder dargestellt werden. Durch das Ausblenden des Carets wird seine aktuelle Form nicht entfernt.

HideCaret blendet das Caret nur dann aus, wenn es dem bezeichneten Fenster zugeordnet ist. Ist der Parameter *Wnd* 0, wird das Caret nur ausgeblendet, wenn ein Fenster in der aktuellen Task das Caret besitzt. Das Ausblenden geschieht kumulativ. Wurde *HideCaret* fünfmal hintereinander aufgerufen wurde, so muß *ShowCaret* fünfmal aufgerufen werden, damit das Caret wieder dargestellt wird.

Wnd: Bezeichnet das Fenster, das das Caret besitzt oder ist 0, um indirekt das Fenster in der aktuellen Task, welches das Caret besitzt, anzugeben.

Querverweis *ShowCaret*

HiliteMenuItem

Syntax
```
function HiliteMenuItem(Wnd: HWnd; Menu: HMenu;
IDHilite, Hilite: Word): Bool;
```

Beschreibung Diese Funktion hebt eine Hauptmenüoption (Menüleiste) hervor oder setzt sie auf den Normalzustand zurück.

Wnd: Bezeichnet das Fenster, welches das Menü enthält.

Menu: Bezeichnet das Hauptmenü, das die hervorzuhebende Option enthält.

IDHilite: Bestimmt den Word-Bezeichner der Menüoption oder deren Offset im Menü in Abhängigkeit vom Parameter *Hilite*.

Hilite: Bestimmt, ob die Menüoption hervorgehoben oder die Hervorhebung aufgehoben werden soll.

Möglich ist eine Kombination von *mf_Hilite* oder *mf_Unhilite* mit *mf_ByCommand* oder *mf_ByPosition*. Diese Werte können mit Hilfe des bitweisen **or**-Operators kombiniert werden. Sie haben die folgende Bedeutung:

Wert	Bedeutung
mf_ByCommand	Interpretiert *IDHiliteItem* als ID der Menüoption (Standardinterpretation)
mf_ByPosition	Interpretiert *IDHiliteItem* als Offset.
mf_Hilite	Stellt die Option hervorgehoben dar. Wenn dieser Wert nicht angegeben ist, wird die Option in den normalen Zustand zurückgesetzt.
mf_UnHilite	Setzt die Option in den normalen Zustand zurück.

Rückgabewert Gibt an, ob die Menüoption als Ergebnis der Funktion hervorgehoben wurde oder nicht. Er ist *True*, wenn das Element in den angegebenen Zustand versetzt wurde, andernfalls *False*.

Hinweis Die Flags *mf_Hilite* und *mf_UnHilite* können nur mit der Funktion *HiliteMenuItem*, nicht aber mit der Funktion *ModifyMenu* verwendet werden.

HiWord

Syntax
```
function HiWord(A: LongInt): Word;
```

Beschreibung Diese Inline-Funktion liefert das höherwertige *Word* des 32-Bit-Integer, der vom Parameter *A* bezeichnet wird.

A: Bestimmt den zu bearbeitenden Wert.

Rückgabewert Der Rückgabewert ist das höherwertige *Word* des gegebenen 32-Bit-Integer.

InflateRect

Syntax
```
procedure InflateRect(var Rect: TRect; X, Y: Integer);
```

Beschreibung Diese Prozedur vergrößert oder verkleinert Höhe und Breite des angegebenen Rechtecks. Sie fügt *X* Einheiten an die linke und die rechte Seite des Rechtecks sowie *Y* Einheiten an die Ober- und Unterseite an. Die Parameter *X* und *Y* sind vorzeichenbehaftete Werte; positive Werte vergrößern die Breite und die Höhe, negative Werte verringern sie.

Rect: TRect-Datenstruktur, die verändert werden soll.

X: Bestimmt den Betrag, um den die Breite des Rechtecks vergrößert oder verkleinert werden soll. Ist *X* negativ, wird die Breite verringert.

Y: Bestimmt den Betrag, um den die Höhe des Rechtecks vergrößert oder verkleinert werden soll. Ist *Y* negativ, wird die Höhe verringert.

Hinweis Die Koordinatenwerte eines Rechtecks dürfen nicht größer als 32767 und nicht kleiner als -32767 Einheiten sein. Die Parameter *X* und *Y* müssen so gewählt werden, daß ungültige Rechtecke vermieden werden.

InitAtomTable

Syntax
```
function InitAtomTable(Size: Integer): Bool;
```

Beschreibung Diese Funktion initialisiert eine Atom-Hash-Tabelle und setzt ihre Größe entsprechend der Vorgabe des Parameters *Size*.

Wird diese Funktion nicht aufgerufen, wird die Größe der Atom-Hash-Tabelle standardmäßig auf 37 gesetzt.

Diese Funktion sollte, wenn sie benötigt wird, vor jeder anderen Atom-Verwaltungsfunktion aufgerufen werden.

Size: Bestimmt die Größe der Atom-Hash-Tabelle in Tabelleneinträgen. Dieser Wert sollte eine Primzahl sein.

Rückgabewert Nach erfolgreicher Ausführung der Funktion wird *True* zurückgeliefert, andernfalls *False*.

Hinweis Benutzt eine Anwendung eine große Zahl von Atomen, so kann sie die erforderliche Zeit für das Hinzufügen eines Atoms zur Atomtabelle oder das Finden eines Atoms in der Tabelle verringern, indem sie die Tabelle vergrößert. Dies erhöht jedoch den Speicherplatz, der erforderlich ist, um die Tabelle zu verwalten.

Die Standardgröße 37 der globalen Atomtabelle kann nicht geändert werden.

InSendMessage

Syntax
```
function InSendMessage: Bool;
```

Beschreibung Diese Funktion bestimmt, ob die aktuelle Fensterfunktion eine Botschaft bearbeitet, die ihr durch Aufruf der Funktion *SendMessage* übergeben wurde.

InSendMessage hat keine Parameter.

Rückgabewert Der Rückgabewert ist *True*, wenn die Fensterfunktion eine Botschaft bearbeitet, die ihr durch *SendMessage* übergeben wurde, andernfalls ist er *False*.

Hinweis Anwendungen benutzen die Funktion *InSendMessage*, um festzustellen, wie mit Fehlern umgegangen werden soll, die bei der Verarbeitung von Botschaften durch ein inaktives Fenster auftreten.

Benutzt beispielsweise das aktive Fenster *SendMessage*, um eine Informationsanforderung an ein anderes Fenster zu senden, so kann das andere Fenster solange nicht aktiv werden, bis es die Steuerung von *SendMessage* zurückerhält. Ein inaktives Fenster kann einen Anwender nur dann von einem Fehler benachrichtigen, wenn es ein Meldungsfenster erstellt.

InsertMenu

Syntax
```
function InsertMenu(Menu: HMenu; Position, Flags,
IDNewItem: Word; NewItem: PChar): Bool;
```

Beschreibung Diese Funktion fügt an der vom Parameter *Position* vorgegebenen Position eine neue Menüoption ein, wobei die anderen

Optionen nach unten verschoben werden. Die Anwendung kann den Status der Menüoption bestimmen, indem sie die Werte im Parameter *Flags* übergibt.

Menu: Bezeichnet das Menü, das verändert werden soll.

Position: Bestimmt die Menüoption, vor der die neue Menüoption eingefügt werden soll. Die Interpretation des Parameters *Position* hängt von der Einstellung des Parameters *Flags* ab.

Flags	Position:
mf_ByPosition	Bestimmt die Position der vorhandenen Menüoption. Die erste Option im Menü steht an Position 0. Wenn *Position* -1 ist, wird die neue Menüoption am Ende des Menüs angefügt.
mf_ByCommand	Bestimmt die Befehls-ID der vorhandenen Menüoption.

Flags: Bestimmt, wie der Parameter *Position* interpretiert wird und informiert über den Status der neuen Menüoption, wenn diese in das Menü eingefügt wird. Er besteht aus einem oder mehreren Werten, die im folgenden Hinweis beschrieben sind.

IDNewItem: Bestimmt die Befehls-ID der neuen Menüoption oder, falls *Flags* auf *mf_Popup* gesetzt ist, das Menü-Handle des Pop-Up-Menüs.

NewItem: Bestimmt den Inhalt der neuen Menüoption. Wenn *Flags* auf *mf_String* gesetzt ist (Voreinstellung), ist *NewItem* ein Zeiger auf einen nullterminierten String, ist es stattdessen auf *mf_Bitmap* gesetzt, enthält *NewItem* ein Bitmap-Handle (*HBitmap*) in seinem niederwertigen *Word*. Wenn *Flags* auf *mf_OwnerDraw* gesetzt ist, bezeichnet *NewItem* einen von der Anwendung gelieferten 32-Bit-Wert, den diese benutzen kann, um weitere der Menüoption zugeordnete Daten zu verwalten. Dieser 32-Bit-Wert steht der Anwendung im Feld *itemData* der Datenstruktur zur Verfügung, auf die der Parameter *Param* der Botschaften *wm_MeasureItem* und *wm_DrawItem* zeigt. Diese Botschaften werden gesendet, wenn die Menüoption erstmals dargestellt oder wenn sie verändert wird.

Rückgabewert Der Rückgabewert ist *True*, wenn die Funktion erfolgreich war, andernfalls ist er *False*.

Hinweis Bei jeder Veränderung eines Menüs (unabhängig davon, ob es sich in einem dargestellten Fenster befindet oder nicht), sollte eine Anwendung *DrawMenuBar* aufrufen.

Jede der folgenden Gruppen enthält Flags, die nicht zusammen benutzt werden sollten:

mf_ByCommand und *mf_ByPosition*
mf_Disabled, mf_Enabled und *mf_Grayed*
mf_Bitmap, mf_String, mf_OwnerDraw und *mf_Separator*
mf_MenuBarBreak und *mf_MenuBreak*
mf_Checked und *mf_Unchecked*

Folgende Flags können im Parameter *Flags* gesetzt werden:

Flag	Bedeutung
mf_Bitmap	Benutzt ein Bitmap als Option. Das niederwertige Word des Parameters *NewItem* enthält das Handle des Bitmaps.
mf_ByCommand	Bestimmt, daß der Parameter *Position* die Nummer der Steuerelement-ID der Menüoption angibt (Voreinstellung).
mf_ByPosition	Bestimmt, daß der Parameter *Position* statt einer ID-Nummer eine zu verändernde Menüoption bezeichnet.
mf_Checked	Plaziert eine Auswahlmarkierung neben der Menüoption. Stellt die Anwendung Auswahlmarkierung-Bitmaps zur Verfügung (siehe auch Funktion *SetMenuItemBitmaps*), bewirkt das Setzen dieses Flags die Darstellung des Bitmaps neben der Menüoption.
mf_Disabled	Sperrt die Menüoption, so daß sie nicht ausgewählt werden kann, versetzt sie jedoch nicht in den nichtaktivierbaren Zustand.
mf_Enabled	Aktiviert die Menüoption, so daß sie ausgewählt werden kann und hebt die Nichtaktivierbarkeit auf.
mf_Grayed	Sperrt die Menüoption, so daß sie nicht ausgewählt werden kann, und versetzt sie in den nichtaktivierbaren Zustand.
mf_MenuBarBreak	Ist gleichbedeutend mit *mf_MenuBreak* mit der Ausnahme, daß bei Pop-Up-Menüs die neue Spalte von der alten Spalte durch eine vertikale Linie getrennt wird.
mf_MenuBreak	Plaziert die Menüoption für statische Menüleistenelemente in einer neuen Zeile. Bei Pop-Up-Menüs wird die Menüoption in einer neuen Spalte ohne Spalten-Trennlinie plaziert.
mf_OwnerDraw	Gibt an, daß die Option eine von der Anwendung gezeichnete Option ist. Das Fenster, dem das Menü gehört, empfängt eine Botschaft *wm_MeasureItem,* wenn das Menü zum ersten Mal dargestellt wird, um Höhe und Breite der Menüoption abzurufen.

Flag	Bedeutung
	Die Botschaft *wm_DrawItem* wird immer dann an die Anwendung gesendet, wenn die Anwendung das Erscheinungsbild der Menüoption aktualisieren muß. Diese Option ist für Hauptmenüoptionen nicht gültig.
mf_Popup	Bestimmt, daß der Menüoption ein Pop-up-Menü zugeordnet ist. Der Parameter *IDNewItem* bezeichnet ein Handle auf ein Pop-Up-Menü, das der Option zugeordnet werden soll. Man sollte das Flag *mf_OwnerDraw* benutzen, um entweder ein Haupt-Pop-Up-Menü oder ein hierarchisches Pop-Up-Menü an eine Pop-up-Menüoption anzufügen.
mf_Separator	Zeichnet eine horizontale Trennlinie. Man kann dieses Flag in einem Pop-up-Menü benutzen. Diese Linie kann nicht gesperrt, hervorgehoben oder in nichtaktivierbaren Zustand versetzt werden. Windows ignoriert die Parameter *NewItem* und *IDNewItem*.
mf_String	Bestimmt, daß die Menüoption ein String ist. Der Parameter *NewItem* zeigt auf den String für die Menüoption.
mf_Unchecked	Plaziert keine Auswahlmarkierung neben der Option (Voreinstellung). Stellt die Anwendung Auswahlmarkierung-Bitmaps zur Verfügung (siehe auch *SetMenuItemBitmaps*), bewirkt das Setzen dieses Flags die Darstellung des Bitmaps neben der Menüoption.

Querverweis *DrawMenuBar, wm_DrawItem, wm_MeasureItem*

IntersectClipRect

Syntax
```
function IntersectClipRect(DC: HDC; X1, Y1, X2, Y2:
Integer): Integer;
```

Beschreibung Erzeugt eine neue Clipping-Region, indem sie den Überschneidungsbereich der aktuellen Region und des von *X1*, *Y1*, *X2* und *Y2* gegebenen Rechtecks bildet. Windows beschneidet alle folgenden Ausgaben so, daß sie in die neuen Grenzen passen.

DC: Bezeichnet den neuen Gerätekontext.

X1: Bestimmt die virtuelle x-Koordinate der oberen linken Ecke des Rechtecks.

Y1: Bestimmt die virtuelle y-Koordinate der oberen linken Ecke des Rechtecks.

X2: Bestimmt die virtuelle x-Koordinate der unteren rechten Ecke des Rechtecks.

Y2: Bestimmt die virtuelle y-Koordinate der unteren rechten Ecke des Rechtecks.

Rückgabewert Der Rückgabewert gibt den Typ der neuen Clipping-Region an. Er kann einer der folgenden Werte sein:

ComplexRegion	Neue Clipping-Region hat überlappende Grenzen.
Error	Gerätekontext ist nicht gültig.
NullRegion	Neue Clipping-Region ist leer.
SimpleRegion	Neue Clipping-Region hat keine überlappenden Grenzen.

Hinweis Die Breite des Rechtecks, gebildet durch die Differenz *X2 - X1*, darf 32767 Einheiten nicht überschreiten. Dieser Grenzwert gilt auch für die Höhe des Rechtecks.

IntersectRect

Syntax
```
function IntersectRect(var DestRect, Src1Rect,
Src2Rect: TRect): Integer;
```

Beschreibung Diese Funktion erzeugt den Überschneidungsbereich von zwei existierenden Rechtecken. Der Überschneidungsbereich ist das größte Rechteck, das in beiden Rechtecken enthalten ist.

IntersectRect kopiert die Koordinaten des neuen Rechtecks in die *TRect*-Datenstruktur, auf die *DestRect* zeigt.

DestRect: TRect-Datenstruktur, die den Überschneidungsbereich aufnehmen soll.

Src1Rect: TRect-Datenstruktur, die ein Quellrechteck enthält.

Src2Rect: TRect-Datenstruktur, die ein Quellrechteck enthält.

Rückgabewert Der Rückgabewert gibt den Überschneidungsbereich von zwei Rechtecken an. Er ist ungleich 0, wenn der Überschneidungsbereich der beiden Rechtecke nicht leer ist, andernfalls 0.

InvalidateRect

Syntax
```
procedure InvalidateRect(Wnd: HWnd; Rect: PRect; Erase:
Bool);
```

Beschreibung Diese Prozedur macht den Client-Bereich innerhalb des gegebenen Rechtecks ungültig, indem dieses an die Aktualisierungsregion von Windows angefügt wird. Das ungültig gemachte Rechteck wird genauso wie alle anderen Bereiche in der Aktualisierungsregion zum Zeichnen gekennzeichnet,

wenn die nächste Botschaft *wm_Paint* eintrifft. Die ungültig gemachten Bereiche sammeln sich in der Aktualisierungsregion solange an, bis die Region beim Eintreffen der nächsten Botschaft *wm_Paint* bearbeitet wird oder die Region mit *ValidateRect* oder *ValidateRgn* auf Gültigkeit überprüft wird.

Der Parameter *Erase* gibt an, ob der Hintergrund innerhalb des Aktualisierungsbereiches gelöscht werden soll, wenn die Aktualisierungsregion bearbeitet wird. Ist *Erase True*, wird der Hintergrund gelöscht, sobald die Funktion *BeginPaint* aufgerufen wird, ist es *False*, bleibt der Hintergrund unverändert. Ist *Erase* für einen Teil der Aktualisierungsregion *True*, wird der Hintergrund der gesamten Region gelöscht, nicht nur im angegebenen Teil.

Wnd: Bezeichnet das Fenster, dessen Aktualisierungsregion verändert werden soll.

Rect: Zeigt auf eine *TRect*-Datenstruktur, die das Rechteck in Client-Koordinaten enthält, das an die Aktualisierungsregion angefügt werden soll. Ist der Parameter *Rect* 0, wird der gesamte Client-Bereich an die Aktualisierungsregion angefügt.

Erase: Bestimmt, ob der Hintergrund innerhalb der Aktualisierungsregion gelöscht werden soll.

Hinweis Windows sendet immer dann eine Botschaft *wm_Paint* an ein Fenster, wenn dessen Aktualisierungsregion nicht leer ist und in der Anwendungs-Warteschlange keine weiteren Botschaften für dieses Fenster vorliegen.

Querverweis *ValidateRect, ValidateRgn, wm_Paint*

InvalidateRgn

Syntax
```
procedure InvalidateRgn(Wnd: HWnd; Rgn: HRgn; Erase:
Bool);
```

Beschreibung Diese Prozedur macht den Client-Bereich innerhalb der angegebenen Region ungültig, indem sie ihn zur Aktualisierungsregion des gegebenen Fensters hinzufügt. Die ungültig gemachte Region wird genau wie alle anderen Bereiche in der Aktualisierungsregion zum Zeichnen gekennzeichnet, wenn die nächste Botschaft *wm_Paint* eintrifft. Die ungültig gemachten Regionen sammeln sich solange in der Aktualisierungsregion an, bis die Region beim nächsten Eintreffen einer Botschaft *wm_Paint* bearbeitet wird oder die Region mit *ValidateRect* oder *ValidateRgn* auf Gültigkeit überprüft wird.

Der Parameter *Erase* bestimmt, ob der Hintergrund innerhalb der Aktualisierungsregion gelöscht werden soll, wenn die Aktualisierungsregion bearbeitet wird. Ist *Erase True*, wird der Hintergrund beim Aufruf der Funktion *BeginPaint* gelöscht, ist

Erase False, bleibt der Hintergrund unverändert. Wenn *Erase* für irgendeinen Teil der Aktualisierungsregion *True* ist, wird der Hintergrund in der gesamten Region gelöscht, nicht nur im angegebenen Teil.

Wnd: Bezeichnet das Fenster, dessen Aktualisierungsregion verändert werden soll.

Rgn: Bezeichnet die Region, die an die Aktualisierungsregion angefügt werden soll. Es wird davon ausgegangen, daß die Region in Client-Koordinaten angegeben ist.

Erase: Bestimmt, ob der Hintergrund innerhalb der Aktualisierungsregion gelöscht werden soll.

Hinweis Windows sendet immer dann eine Botschaft *wm_Paint* an ein Fenster, wenn seine Aktualisierungsregion nicht leer ist und es in der Anwendungs-Warteschlange keine anderen Botschaften für das Fenster gibt. Die angegebene Region muß vorher mit einer entsprechenden Funktion erzeugt worden sein.

Querverweis *ValidateRect, ValidateRgn, wm_Paint*

InvertRect

Syntax
```
procedure InvertRect(DC: HDC; var Rect: TRect);
```

Beschreibung Diese Prozedur invertiert die Inhalte des gegebenen Rechtecks. Auf Monochrombildschirmen färbt sie weiße Pixel schwarz und schwarze Pixel weiß. Auf Farbbildschirmen hängt die Inversion davon ab, wie Farben für den Bildschirm erzeugt werden. Der zweimalige Aufruf von *InvertRect* für dasselbe Rechteck stellt den Bildschirm mit den vorherigen Farben wieder her.

DC: Bezeichnet den Gerätekontext.

Rect: Eine *TRect*-Datenstruktur, die die virtuellen Koordinaten des zu invertierenden Rechtecks enthält.

Hinweis Die Funktion vergleicht die Werte der Felder *top*, *bottom*, *left* und *right* des bezeichneten Rechtecks. Wenn *bottom* kleiner oder gleich *top* ist oder *right* kleiner oder gleich *left* ist, wird das Rechteck nicht invertiert.

InvertRgn

Syntax
```
function InvertRgn(DC: HDC; Rng: HRng): Bool;
```

Beschreibung Diese Funktion invertiert die Farben in der Region, die der Parameter *Rgn* bezeichnet. Auf Monochrombildschirmen färbt die Funktion *InvertRgn* weiße Pixel schwarz und schwarze Pixel

weiß. Auf Farbbildschirmen hängt die Inversion davon ab, wie Farben für den Bildschirm erzeugt werden.

DC: Bezeichnet den Gerätekontext.

Rgn: Bezeichnet die Region, die invertiert werden soll. Die Koordinaten für die Region sind in gerätespezifischen Einheiten angegeben.

Rückgabewert Der Rückgabewert ist *True*, wenn die Funktion erfolgreich war, andernfalls ist er *False*.

IsCharAlpha

Syntax
```
function IsCharAlpha(Chr: Char): Bool;
```

Beschreibung Diese Funktion stellt fest, ob ein Zeichen ein alphabetisches Zeichen ist. Diese Feststellung wird vom Sprachtreiber getroffen, der vom Anwender im Setup oder in der Systemsteuerung ausgewählt wurde.

Chr: Bestimmt das Zeichen, das überprüft werden soll.

Rückgabewert Der Rückgabewert ist *True*, wenn das Zeichen alphabetisch ist, andernfalls ist er *False*.

IsCharAlphaNumeric

Syntax
```
function IsCharAlphaNumeric(Chr: Char): Bool;
```

Beschreibung Diese Funktion stellt fest, ob ein Zeichen ein alphanumerisches Zeichen ist. Diese Feststellung wird vom Sprachtreiber getroffen, der vom Anwender im Setup oder in der Systemsteuerung ausgewählt wurde.

Chr: Bestimmt das Zeichen, das überprüft werden soll.

Rückgabewert Der Rückgabewert ist *True*, wenn das Zeichen alphanumerisch ist, andernfalls ist er *False*.

IsCharLower

Syntax
```
function IsCharLower(Chr: Char): Bool;
```

Beschreibung Diese Funktion stellt fest, ob das Zeichen in Kleinschreibung vorliegt. Diese Feststellung wird vom Sprachtreiber getroffen, der vom Anwender im Setup oder in der Systemsteuerung ausgewählt wurde.

Chr: Bestimmt das Zeichen, das überprüft werden soll.

Rückgabewert Der Rückgabewert ist *True*, wenn das Zeichen in Kleinschreibung vorliegt, andernfalls ist er *False*.

IsCharUpper

Syntax
```
function IsCharUpper(Chr:Char): Bool;
```

Beschreibung Diese Funktion stellt fest, ob das Zeichen in Großschreibung vorliegt. Diese Feststellung wird vom Sprachtreiber getroffen, der vom Anwender im Setup oder in der Systemsteuerung ausgewählt wurde.

Chr: Bestimmt das Zeichen, das überprüft werden soll.

Rückgabewert Der Rückgabewert ist *True*, wenn das Zeichen in Großschreibung vorliegt, andernfalls ist er *False*.

IsChild

Syntax
```
function IsChild(WndParent, Wnd: HWnd): Bool;
```

Beschreibung Diese Funktion zeigt an, ob das vom Parameter *Wnd* angegebene Fenster ein untergeordnetes Fenster oder ein anderer direkter Abkömmling des Fensters ist, das der Parameter *WndParent* bezeichnet.

Ein untergeordnetes Fenster ist der direkte Abkömmling des angegebenen übergeordneten Fensters, wenn dieses übergeordnete Fenster in der Kette der übergeordneten Fenster steht, die vom ursprünglichen Pop-Up-Fenster zum untergeordneten Fenster führt.

WndParent: Bezeichnet ein Fenster.

Wnd: Bezeichnet das Fenster, das überprüft werden soll.

Rückgabewert Der Rückgabewert ist *True*, wenn das vom Parameter *Wnd* bezeichnete Fenster ein untergeordnetes Fenster des vom Parameter *WndParent* bezeichneten Fensters ist, andernfalls ist er *False*.

IsClipboardFormatAvailable

Syntax
```
function IsClipboardFormatAvailable(Format: Word): Bool;
```

Beschreibung Diese Funktion ermittelt, ob Daten eines bestimmten Typs in der Zwischenablage existieren.

Format: Bestimmt ein registriertes Zwischenablageformat. Informationen über Zwischenablageformate finden Sie im Abschnitt »cf_Xxxx« in Kapitel 1 »Windows-Stile und -Konstanten«.

Rückgabewert Der Rückgabewert ist *True*, wenn Daten, im bezeichneten Format vorliegen, andernfalls ist er *False*.

Hinweis Diese Funktion wird normalerweise während der Verarbeitung der Botschaften *wm_InitMenu* oder *wm_InitMenuPopup* verwendet, um festzustellen, ob die Zwischenablage Daten enthält, die in die Anwendung eingefügt werden können. Sind solche Daten vorhanden, versetzt die Anwendung üblicherweise den Befehl *Paste* (Kopieren) in ihrem Edit-Menü (Bearbeiten) in den aktivierbaren Zustand.

Querverweis *SetClipboardData*

IsDialogMessage

Syntax
```
function IsDialogMessage(Dlg: HWnd; var Msg: TMsg):
Bool;
```

Beschreibung Diese Funktion ermittelt, ob die angegebene Botschaft für das durch *Dlg* bezeichnete nichtmodale Dialogfenster bestimmt ist und verarbeitet die Botschaft in diesem Fall automatisch.

Wenn *IsDialogMessage* eine Botschaft verarbeitet, prüft sie auf Tastaturbotschaften und konvertiert sie in Auswahlbefehle für das zugehörige Dialogfenster. Z.B. wählt die Tabulatortaste das nächste Steuerelement oder Gruppe von Steuerelementen aus. Die Pfeiltaste nach unten wählt das nächste Steuerelement in einer Gruppe aus. Wenn eine Botschaft von *IsDialogMessage* verarbeitet wurde, darf sie nicht an die Funktionen *TranslateMessage* oder *DispatchMessage* übergeben werden, da *IsDialogMessage* die gesamte notwendige Übersetzung und Verteilung vornimmt. *IsDialogMessage* sendet eine Botschaft *wm_GetDlgCode* an die Dialogfunktion, um zu bestimmen, welche Tasten verarbeitet werden sollen.

Dlg: Bezeichnet das Dialogfenster.

Msg: TMsg-Datenstruktur, die die Botschaft enthält, die geprüft werden soll.

Rückgabewert Der Rückgabewert ist *True*, wenn die Botschaft verarbeitet worden ist, andernfalls ist er *False*.

Hinweis Obwohl *IsDialogMessage* für nichtmodale Dialogfenster vorgesehen ist, kann diese Funktion für jedes Fenster benutzt werden, das Steuerelemente enthält, um dieselbe Tastaturauswahl wie in einem Dialogfenster zu ermöglichen.

IsDlgButtonChecked

Syntax
```
function IsDlgButtonChecked(Wnd: HWnd; IDButton:
Integer): Word;
```

Beschreibung Diese Funktion stellt fest, ob ein Schalter eine Auswahlmarkierung hat und ob ein dreistufiger Schalter nichtaktivierbar,

ausgewählt oder keines von beiden ist. Die Funktion *IsDlgButtonChecked* übergibt eine Botschaft *bm_GetCheck* an den Schalter.

Wnd: Bezeichnet das Dialogfenster, das den Schalter enthält.

IDButton: Bestimmt den Integer-Bezeichner des Schalters.

Rückgabewert Der Rückgabewert ist ungleich 0, wenn der angegebene Schalter markiert ist, andernfalls ist er 0.

Bei dreistufigen Schaltern ist der Rückgabewert 2, wenn der Schalter nichtaktivierbar ist, er ist 1, wenn der Schalter markiert ist und 0, wenn er nicht markiert ist.

IsIconic

Syntax `function IsIconic(Wnd: HWnd): Bool;`

Beschreibung Diese Funktion ermittelt, ob ein Fenster zum Symbol verkleinert ist.

Wnd: Bezeichnet das Fenster.

Rückgabewert Der Rückgabewert ist *True*, wenn das Fenster verkleinert wurde, andernfalls *False*.

IsRectEmpty

Syntax `function IsRectEmpty(var Rect: TRect): Bool;`

Beschreibung Diese Funktion stellt fest, ob das bezeichnete Rechteck leer ist oder nicht. Ein Rechteck ist leer, wenn Höhe und/oder Breite 0 sind.

Rect: TRect-Datenstruktur, die das angegebene Rechteck enthält.

Rückgabewert Der Rückgabewert ist *True*, wenn das Rechteck leer ist, andernfalls *False*.

IsWindow

Syntax `function IsWindow(Wnd: HWnd): Bool;`

Beschreibung Diese Funktion stellt fest, ob das Fenster, das durch den Parameter *Wnd* bezeichnet ist, ein gültiges existierendes Fenster ist.

Wnd: Bezeichnet das Fenster.

Rückgabewert Der Rückgabewert ist ungleich 0, wenn *hWnd* ein gültiges Fenster ist, andernfalls 0.

IsWindowEnabled

Syntax
```
function IsWindowEnabled(Wnd: HWnd): Bool;
```

Beschreibung Diese Funktion ermittelt, ob das durch *Wnd* angegebene Fenster zur Entgegennahme von Maus- und Tastatureingaben befähigt ist.

Wnd: Bezeichnet das Fenster.

Rückgabewert Der Rückgabewert ist *True*, wenn das Fenster aktiviert ist, andernfalls *False*.

Hinweis Ein untergeordnetes Fenster empfängt nur Eingaben, wenn es sowohl aktiviert als auch sichtbar ist.

IsWindowVisible

Syntax
```
function IsWindowVisible(Wnd: HWnd): Bool;
```

Beschreibung Diese Funktion gibt *True* zurück, wenn ein Fenster mit der Funktion *ShowWindow* sichtbar gemacht wurde. Auch wenn das bezeichnete Fenster vollständig von einem anderen untergeordneten oder Pop-Up-Fenster verdeckt ist, ist der Rückgabewert *True*.

Wnd: Bezeichnet das Fenster.

Rückgabewert Der Rückgabewert ist *True*, wenn das gegebene Fenster auf dem Bildschirm vorhanden ist, andernfalls ist er *False*.

IsZoomed

Syntax
```
function IsZoomed(Wnd: HWnd): Bool;
```

Beschreibung Diese Funktion stellt fest, ob ein Fenster vergrößert ist oder nicht.

Wnd: Bezeichnet das Fenster.

Rückgabewert Der Rückgabewert ist *True*, wenn das Fenster vergrößert ist, andernfalls *False*.

KillTimer

Syntax
```
function KillTimer(Wnd: HWnd; IDEvent: Integer): Bool;
```

Beschreibung Diese Funktion eliminiert das Timer-Ereignis, das durch die Parameter *Wnd* und *IDEvent* bezeichnet wird.

Alle anstehenden *wm_Timer*- Botschaften, die mit dem Timer verbunden sind, werden aus der Botschafts-Warteschlange entfernt.

Wnd: Bezeichnet das Fenster, das mit dem angegebenen Timer-Ereignis verbunden ist. Dies muß derselbe Wert sein, der beim Aufruf der Funktion *SetTimer* als Parameter *Wnd* übergeben wurde, wenn diese das Timer-Ereignis erzeugt hat.

IDEvent: Bestimmt das Timer-Ereignis, das eliminiert werden soll. Wenn die Anwendung *SetTimer* mit auf 0 gesetztem Parameter *Wnd* aufgerufen hat, muß es der Ereignisbezeichner sein, der von *SetTimer* zurückgegeben wurde. Wenn der Parameter *Wnd* von *SetTimer* ein gültiges Fenster-Handle war, muß *IDEvent* der Wert des Parameters *IDEvent* sein, der an *SetTimer* übergeben wurde.

Return Value Der Rückgabewert ist *True*, wenn das Ereignis eliminiert wurde und *False*, wenn die Funktion das angegebene Timer-Ereignis nicht finden konnte.

Querverweis *SetTimer*

_lclose

Syntax
```
function _lclose(FileHandle: Integer): Integer;
```

Beschreibung Diese Funktion schließt die durch den Parameter *FileHandle* angegebene Datei. Diese Datei ist dann nicht länger für Lese- oder Schreiboperationen verfügbar. Das Argument *FileHandle* wird durch den Funktionsaufruf zurückgeliefert, der diese Datei erzeugt oder zuletzt geöffnet hat.

FileHandle: Bestimmt das DOS-Datei-Handle der Datei, die geschlossen werden soll.

Rückgabewert Der Rückgabewert ist 0, wenn die Funktion die Datei geschlossen hat, im Fall eines Fehlers -1.

_lcreat

Syntax
```
function _lcreat(PathName: PChar; Atribute: Integer):
Integer;
```

Beschreibung Diese Funktion öffnet eine Datei mit dem durch den Parameter *PathName* angegebenen Namen. Der Parameter *Atribute* bestimmt das Attribut der Datei. Wenn keine Datei mit dem angegebenen Namen existiert, erzeugt die Funktion eine neue Datei und öffnet sie für Schreiboperationen. Beim Öffnen der Datei durch *_lcreat* wird der Dateizeiger auf den Anfang der Datei gesetzt.

PathName: Zeigt auf einen nullterminierten String, der den Namen der zu öffnenden Datei enthält. Der String muß aus Zeichen des ANSI-Zeichensatzes bestehen.

Atribute: Bestimmt das Dateiattribut. Der Parameter muß einen der folgenden Werte annehmen:

Wert	Bedeutung
0	Kann ohne Einschränkung gelesen oder beschrieben werden.
1	Schreibgeschützt; kann nicht zum Schreiben geöffnet werden; eine Datei gleichen Namens kann nicht erzeugt werden.
2	Unsichtbar; wird bei der Verzeichnissuche nicht angezeigt.
3	System; wird bei der Verzeichnissuche nicht angezeigt.

Rückgabewert Der Rückgabewert gibt ein MS-DOS-Datei-Handle an, wenn die Funktion erfolgreich ausgeführt wurde, andernfalls wird -1 zurückgeliefert.

LimitEmsPages

Syntax
```
procedure LimitEmsPages (Kbytes: LongInt);
```

Beschreibung Diese Prozedur begrenzt den Umfang des Expanded Memory, den Windows einer Anwendung zuweisen wird. Der Umfang des Expanded Memory, den die Anwendung durch direkten Aufruf des INT 67H ansprechen kann, wird nicht begrenzt.

Kbytes: Gibt die Anzahl KByte des Expanded Memory an, auf die die Anwendung zugreifen kann.

Hinweis *LimitEmsPages* hat nur dann ein Resultat, wenn Expanded Memory installiert ist und von Windows genutzt wird. Benutzt Windows das Expanded Memory nicht, hat die Funktion keine Wirkung.

LineDDA

Syntax
```
procedure LineDDA(X1, Y1, X2, Y2: Integer; LineFunc:
TFarProc; Data: Pointer);
```

Beschreibung Diese Prozedur berechnet alle aufeinander folgenden Punkte einer Linie vom Anfangspunkt (*X1*, *Y1*) bis zum Endpunkt *(X2*, *Y2*). Der Endpunkt ist nicht als Teil der Linie eingeschlossen. Für jeden Punkt der Linie ruft die Funktion *LineDDA* die anwendungsunterstützte Funktion auf, auf die der Parameter *LineFunc* zeigt und übergibt dieser Funktion die Koordinaten des aktuellen Punktes und den Parameter *Data*.

X1: Bestimmt die virtuelle x-Koordinate des ersten Punktes.

Y1: Bestimmt die virtuelle y-Koordinate des ersten Punktes.

X2: Bestimmt die virtuelle x-Koordinate des Endpunktes.

Y2: Bestimmt die virtuelle y-Koordinate des Endpunktes.

LineFunc: Adresse der Prozedurinstanz der Callback-Funktion.

Data: Zeigt auf die Daten für die Callback-Funktion.

Hinweis

Die Adresse, die mit dem Parameter *LineFunc* übergeben wird, muß vorab durch den Aufruf der Funktion *MakeProcInstance* erzeugt werden.

Callback-Funktion

```
procedure LineFunc(X, Y: Integer; lpData: PChar); export
```

LineFunc: Ein Platzhalter für den Namen der anwendungsunterstützten Funktion.

Parameter	Definition
X	Gibt die X-Koordinate des aktuellen Punktes an.
Y	Gibt die Y-Koordinate des aktuellen Punktes an.
lpData	Zeigt auf die Anwendungsdaten.

LineTo

Syntax

```
function LineTo(DC: HDC; X, Y: Integer): Bool;
```

Beschreibung

Diese Funktion zeichnet eine Linie von der aktuellen Position bis zum Punkt, der durch die Parameter *X* und *Y* angegeben wird. Der Endpunkt ist nicht in der Linie enthalten. Die Linie wird mit dem ausgewählten Stift gezeichnet. Wenn kein Fehler auftritt, wird die aktuelle Position auf (*X*, *Y*) gesetzt.

DC: Bezeichnet den Gerätekontext.

X: Bestimmt die virtuelle x-Koordinate des Endpunkts der Linie.

Y: Bestimmt die virtuelle y-Koordinate des Endpunkts der Linie.

Rückgabewert

Der Rückgabewert ist *True*, wenn die Linie gezeichnet wurde, andernfalls *False*.

_llseek

Syntax

```
function _llseek(FileHandle: Integer; Offset: LongInt;
Origin: Integer): LongInt;
```

Beschreibung

Diese Funktion positioniert den Zeiger in einer vorher geöffneten Datei neu. Der Parameter *Origin* legt die Startposition in der Datei fest und *Offset* gibt an, wie weit (in Bytes) die Funktion den Zeiger verschieben soll.

FileHandle: Bestimmt das DOS-Datei-Handle der Datei.

Offset: Bestimmt die Anzahl Bytes, um die der Zeiger verschoben werden soll.

Origin: Bestimmt Startposition und Richtung der Verschiebung.

Der Parameter muß einen der in folgender Tabelle aufgeführten Werte annehmen:

Wert	Bedeutung
0	Bewegt den Zeiger *Offset* Bytes, ausgehend vom Anfang der Datei.
1	Bewegt den Zeiger *Offset* Bytes, ausgehend von der aktuellen Position in der Datei.
2	Bewegt den Zeiger rückwärts *Offset* Bytes, ausgehend vom Ende der Datei.

Rückgabewert Bei erfolgreicher Ausführung wird der neue Offset des Zeigers (in Bytes), ausgehend vom Anfang der Datei, zurückgegeben, andernfalls -1.

Hinweis Beim Öffnen einer Datei wird der Dateizeiger auf den Anfang der Datei positioniert.

Die Funktion *_llseek* erlaubt direkten Zugriff auf den Inhalt einer Datei, indem der Zeiger um einen vorgegebenen beliebigen Betrag bewegt wird, ohne Daten zu lesen.

LoadAccelerators

Syntax
```
function LoadAccelerators(Instance: THandle; TableName:
PChar): THandle;
```

Beschreibung Diese Funktion lädt die durch den Parameter *TableName* benannte Accelerator-Tabelle aus der ausführbaren Datei, die dem durch *Instance* angegebenen Modul zugeordnet ist.

Die Funktion *LoadAccelerators* lädt die Tabelle nur, wenn sie nicht schon vorher geladen wurde, andernfalls vergibt sie ein Handle zur existierenden Tabelle.

Instance: Bezeichnet eine Instanz des Moduls, dessen ausführbare Datei die Accelerator-Tabelle enthält.

TableName: Zeigt auf einen nullterminierten String, der den Namen der Accelerator-Tabelle enthält.

Rückgabewert Wenn die Funktion erfolgreich ausgeführt wurde, bezeichnet der Rückgabewert die geladene Accelerator-Tabelle, ansonsten ist er 0.

LoadBitmap

Syntax
```
function LoadBitmap(Instance: THandle; BitmapName:
PChar): HBitmap;
```

Beschreibung Diese Funktion lädt die durch *BitmapName* benannte Bitmap-Ressource aus der ausführbaren Datei, die dem durch *Instance* angegebenen Modul zugeordnet ist.

Instance: Bezeichnet die Instanz des Moduls, dessen ausführbare Datei das Bitmap enthält.

BitmapName: Zeigt auf einen nullterminierten String, der den Namen des Bitmap enthält.

Rückgabewert Der Rückgabewert bezeichnet das angegebene Bitmap. Er ist 0, wenn dieses Bitmap nicht existiert.

Hinweis Eine Anwendung muß die Funktion *DeleteObject* aufrufen, um jedes Bitmap-Handle zu löschen, das von der Funktion *LoadBitmap* zurückgeliefert wurde. Das gleiche gilt für die vordefinierten Bitmaps, die nachfolgend beschrieben werden.

Die Funktion *LoadBitmap* kann auch benutzt werden, um auf die vordefinierten Bitmaps, die von Windows benutzt werden, zuzugreifen. Der Parameter *Instance* muß auf 0 gesetzt werden und der Parameter *BitmapName* muß einen der folgenden Werte annehmen:

obm_BtnConers	*obm_Old_Restore*
obm_BtSize	*obm_Old_RgArrow*
obm_Check	*obm_Old_UpArrow*
obm_CheckBoxes	*obm_Old_Zoom*
obm_Close	*obm_Reduce*
obm_Combo	*obm_Reduced*
obm_DnArrow	*obm_Restore*
obm_DnArrowD	*obm_Restored*
obm_LfArrow	*obm_RgArrow*
obm_LfArrowD	*obm_RgArrowD*
obm_MnArrow	*obm_Size*
obm_Old_Close	*obm_UpArrow*
obm_Old_DnArrow	*obm_UpArrowD*
obm_Old_LfArrow	*obm_Zoom*
obm_Old_Reduce	*obm_ZoomD*

Bitmap-Namen, die mit *obm_Old* beginnen, bezeichnen Bitmaps, die von Windows-Versionen vor der Version 3.0 benutzt werden. Der Parameter *BitmapName* kann auch einen Wert vom Typ *MakeIntResource* beinhalten. Ist dies der Fall, muß die ID im niederwertigen Word von *BitmapName* stehen und das höherwertige Word auf 0 gesetzt werden.

LoadCursor

Syntax

```
function LoadCursor(Instance: THandle; CursorName:
PChar): HCursor;
```

Beschreibung Diese Funktion lädt die durch den Parameter *CursorName* benannte Mauszeiger-Ressource aus der ausführbaren Datei, die dem durch den Parameter *Instance* angegebenen Modul zugeordnet ist.

Die Funktion lädt den Mauszeiger nur in den Speicher, wenn er nicht schon vorher geladen wurde, andernfalls vergibt sie ein Handle zur existierenden Ressource.

Instance: Bezeichnet die Instanz des Moduls, dessen ausführbare Datei den Mauszeiger enthält.

CursorName: Zeigt auf einen nullterminierten String, der den Namen der Mauszeiger-Ressource enthält.

Rückgabewert Bei erfolgreicher Ausführung bezeichnet der Rückgabewert den neugeladenen Mauszeiger, andernfalls ist er 0.

Hinweis Die Funktion *LoadCursor* liefert nur ein gültiges Mauszeiger-Handle, wenn der Parameter *CursorName* eine Mauszeiger-Ressource bezeichnet. Bezeichnet *CursorName* eine andere Art von Ressource (z.B. ein Symbol), ist der Rückgabewert nicht 0, obwohl er kein gültiges Mauszeiger-Handle ist.

LoadCursor kann ebenfalls verwendet werden, um auf einen von Windows verwendeten, vordefinierten Mauszeiger zuzugreifen. Dazu muß der Parameter *Instance* auf 0 gesetzt sein und der Parameter *CursorName* muß einen der folgenden Werte annehmen:

Wert	Bedeutung
idc_Arrow	Standardpfeilzeiger.
idc_Cross	Fadenkreuzzeiger.
idc_IBeam	I-Balkenzeiger in Texten.
idc_Icon	Leeres Symbol.
idc_Size	Vierpfeilzeiger. Die Pfeile zeigen nach Norden, *Osten, Süden und Westen.*
idc_SizeNESW	*Doppelpfeilzeiger. Die Pfeile zeigen nach Nordost und Südwest.*
idc_SizeNS	Doppelpfeilzeiger. Die Pfeile zeigen nach Norden und Süden.
idc_SizeNWSE	Doppelpfeilzeiger. Die Pfeile zeigen nach Nordwest und Südost.
idc_SizeWE	Doppelpfeilzeiger. Die Pfeile zeigen nach Westen und Osten.
idc_UpArrow	Vertikaler Pfeilzeiger nach oben.
idc_Wait	Sanduhrzeiger.

Der Parameter *CursorName* kann auch einen Wert vom Typ *MakeIntResource* enthalten. Ist dies der Fall, muß die ID im niederwertigen Word von *Cursorame* stehen und das höherwertige Word muß auf 0 gesetzt werden.

LoadIcon

Syntax
```
function LoadIcon(Instance: THandle; IconName: PChar): HIcon;
```

Beschreibung Diese Funktion lädt die durch den Parameter *IconName* benannte Symbol-Ressource aus der ausführbaren Datei, die dem durch den Parameter *Instance* bezeichneten Modul zugeordnet ist.

Die Funktion lädt das Symbol nur, wenn es nicht schon vorher geladen wurde, andernfalls vergibt sie ein Handle zur existierenden Ressource.

Instance: Bezeichnet die Instanz des Moduls, dessen ausführbare Datei das Symbol enthält.

IconName: Zeigt auf einen nullterminierten String, der den Namen der Symbol-Ressource enthält.

Rückgabewert Bei erfolgreicher Ausführung bezeichnet der Rückgabewert eine Symbol-Ressource, andernfalls ist er 0.

Hinweis Verwenden Sie die Funktion *LoadIcon*, um auf die von Windows benutzten vordefinierten Symbole zuzugreifen. Dazu muß der Parameter *Instance* auf 0 gesetzt werden und der Parameter *IconName* einen der folgenden Werte annehmen:

Wert	Bedeutung
idi_Application	Voreingestelltes Anwendungssymbol.
idi_Asterisk	Stern (verwendet bei Informationsmeldungen).
idi_Exclamation	Ausrufezeichen (verwendet bei Warnmeldungen).
idi_Hand	Symbol in Handform (verwendet bei ernsten Warnmeldungen).
idi_Question	Fragezeichen (verwendet bei Anwenderabfragen).

Der Parameter *IconName* kann auch einen Wert vom Typ *MakeIntResource* enthalten. Ist dies der Fall, muß die ID im niederwertigen Word von *IconName* stehen und das höherwertige Word auf 0 gesetzt werden.

LoadLibrary

Syntax
```
function LoadLibrary(LibFileName: PChar): THandle;
```

Beschreibung Diese Funktion lädt das Bibliotheksmodul, das in der angegebenen Datei enthalten ist und liefert ein Handle zur Instanz des geladenen Moduls.

LibFileName: Zeigt auf einen nullterminierten String, der den Namen der Bibliotheksdatei enthält.

Rückgabewert Der Rückgabewert bezeichnet bei erfolgreicher Ausführung die Instanz des geladenen Bibliothekmoduls, andernfalls hat er einen Wert kleiner als 32, der den Fehler bezeichnet. Folgende Fehlerwerte können zurückgeliefert werden:

Wert	Bedeutung
0	Zu wenig Speicher.
2	Datei nicht gefunden.
3	Pfad nicht gefunden.
5	Versuch, eine Task dynamisch einzubinden.
6	Bibliothek benötigt getrennte Datensegmente für jede Task.
10	Falsche Windows-Version.
11	Ungültige .EXE Datei (Keine Windows .EXE oder Fehler in der .EXE).
12	OS/2 Anwendung.
13	DOS 4.0 Anwendung.
14	Unbekannter .EXE-Typ.
15	Versuch, im Protected Mode (Standard- oder erweiterter 386-Modus) eine .EXE Datei zu laden, die für eine frühere Windows-Version erstellt wurde.
16	Versuch, eine zweite Instanz einer .EXE-Datei zu laden, die mehrere beschreibbare Datensegmente enthält.
17	Versuch, im Large-Frame-EMS-Modus eine zweite Instanz einer Anwendung zu laden, die bestimmte DLLs einbindet, die schon im Gebrauch sind, aber nicht gemeinsam genutzt werden können.
18	Versuch, im Real Mode eine Anwendung zu laden, die nur für den Protected Mode eingerichtet ist.

LoadMenu

Syntax
```
function LoadMenu(Instance: THandle; MenuName: PChar):
HMenu;
```

Beschreibung Diese Funktion lädt die durch den Parameter *MenuName* benannte Menü-Ressource aus der ausführbaren Datei, die dem durch den Parameter *Instance* bezeichneten Modul zugeordnet ist.

Instance: Bezeichnet die Instanz des Moduls, dessen ausführbare Datei das Menü enthält.

MenuName: Zeigt auf einen nullterminierten String, der den Namen der Menü-Ressource enthält.

Rückgabewert Der Rückgabewert bezeichnet eine Menü-Ressource, wenn die Funktion erfolgreich war, andernfalls ist er 0.

Hinweis Der Parameter *MenuName* kann auch einen Wert vom Typ *MakeIntResource* beinhalten. Ist dies der Fall, muß die ID im

niederwertigen Word von *MenuName* stehen und das höherwertige Word muß auf 0 gesetzt werden.

LoadMenuIndirect

Syntax
```
function LoadMenuIndirect(MenuTemplate: Pointer): HMenu;
```

Beschreibung Diese Funktion lädt das Menü in den durch den Parameter *MenuTemplate* benannten Speicher.

Die Vorlage, die von *MenuTemplate* angegeben wird, ist eine Kopfzeile, auf die eine Sammlung einer oder mehrerer *TMenuItemTemplate*-Strukturen folgt, von denen jede einen oder mehrere Menüpunkte oder Pop-Up-Menüs beinhalten kann.

MenuTemplate: Zeigt auf eine Menüvorlage (die eine Sammlung einer oder mehrerer *TMenuItemTemplate*-Strukturen ist).

Rückgabewert Der Rückgabewert bezeichnet das Menü, wenn die Funktion erfolgreich war, andernfalls ist er 0.

LoadModule

Syntax
```
function LoadModule(ModuleName: PChar; ParameterBlock:
Pointer): THandle;
```

Beschreibung Diese Funktion lädt ein Windows-Programm und führt es aus oder erzeugt eine neue Instanz eines existierenden Windows-Programms.

ModuleName: Zeigt auf einen nullterminierten String, der den Dateinamen der Anwendung enthält, die gestartet werden soll. Enthält *ModuleName* keinen Verzeichnispfad, sucht Windows nach der ausführbaren Datei in folgender Reihenfolge:

1. Im aktuellen Verzeichnis.

2. Im Windows-Verzeichnis (das Verzeichnis, in dem sich die Datei WIN.COM befindet); die Funktion *GetWindowsDirectory* liefert den Pfadnamen dieses Verzeichnisses.

3. Im Windows-Systemverzeichnis (das Verzeichnis, das Systemdateien wie KERNEL.EXE enthält); die Prozedur *GetSystemDirectory* liefert den Pfadnamen dieses Verzeichnisses.

4. In den Verzeichnissen, die in der Environment-Variablen PATH aufgelistet sind.

5. Die Liste von Verzeichnissen, die in einem Netz im Befehl MAP angegeben wurden.

Wenn der Dateiname der Anwendung keine Namenserweiterung enthält, wird von .EXE ausgegangen.

ParameterBlock: Zeigt auf eine Datenstruktur, die aus vier Feldern besteht, die einen Parameterblock *definieren*. Die Datenstruktur besteht aus den folgenden Feldern:

Feld	Beschreibung
EnvSeg	Bestimmt die Segmentadresse der Umgebung, unter der das Modul laufen soll; 0 zeigt an, daß die Windows-Umgebung kopiert werden soll.
CmdLine	Zeigt auf einen nullterminierten String, der eine korrekt gebildete Kommandozeile enthält. Dieser String darf eine Länge von 120 Bytes nicht überschreiten.
CmdShow	Zeigt auf eine Datenstruktur, die zwei Werte in Word-Länge enthält. Der erste Wert muß immer auf zwei gesetzt werden. Der zweite Wert gibt an, wie das Anwendungsfenster dargestellt werden soll. In Kapitel 1 »Windows-Stile und -Konstanten« im Abschnitt »sw_Xxxx« finden Sie eine Beschreibung der möglichen Werte.
dwReserved	Ist reserviert und muß 0 sein. Alle ungenutzten Felder sollten auf 0 gesetzt werden, mit Ausnahme von *CmdLine*, das auf einen Nullstring zeigen muß, wenn es nicht benötigt wird.

Rückgabewert

Der Rückgabewert bezeichnet bei erfolgreicher Ausführung der Funktion die Instanz des geladenen Moduls. Andernfalls hat er einen Wert kleiner als 32, der den Fehler bezeichnet. Die folgende Liste beschreibt die Fehlerwerte, die von dieser Funktion zurückgeliefert werden:

Wert	Bedeutung
0	Zu wenig Speicher.
2	Datei nicht gefunden.
3	Pfad nicht gefunden.
5	Versuch, eine Task dynamisch einzubinden.
6	Bibliothek benötigt getrennte Datensegmente für jede Task.
10	Falsche Windows-Version.
11	Ungültige .EXE Datei (Keine Windows-.EXE oder Fehler in der .EXE).
12	OS/2 Anwendung.
13	DOS 4.0 Anwendung.
14	Unbekannter .EXE-Typ.
15	Versuch, im Protected Mode (Standard- oder erweiterter 386-Modus) eine .EXE Datei zu laden, die für eine frühere Windows-Version erstellt wurde.
16	Versuch, eine zweite Instanz einer .EXE-Datei zu laden, die mehrere beschreibbare Datensegmente enthält.

Wert	Bedeutung
17	Versuch, im Large-Frame-EMS-Modus eine zweite Instanz einer Anwendung zu laden, die bestimmte DLLs (dynamische Linkbibliotheken) einbindet, die schon im Gebrauch sind, aber nicht gemeinsam genutzt werden können.
18	Versuch, im Real Mode eine Anwendung zu laden, die nur für den Protected Mode eingerichtet ist.

Hinweis Die Funktion *WinExec* stellt eine alternative Methode dar, ein Programm auszuführen.

Querverweis *WinExec*

LoadResource

Syntax
```
function LoadResource(Instance, ResInfo: THandle):
THandle;
```

Beschreibung Diese Funktion lädt eine durch den Parameter *ResInfo* bezeichnete Ressource aus der ausführbaren Datei, die dem durch den Parameter *Instance* bezeichneten Modul zugeordnet ist.

LoadResource lädt die Ressource nur in den Speicher, wenn sie nicht schon vorher geladen wurde. Andernfalls liefert sie ein Handle zur bestehenden Ressource.

Instance: Bezeichnet eine Instanz des Moduls, dessen ausführbare Datei die Ressource enthält.

ResInfo: Bezeichnet die gewünschte Ressource. Voraussetzung ist, daß dieses Handle durch die Funktion *FindResource* erzeugt wurde.

Rückgabewert Der Rückgabewert bezeichnet den globalen Speicherblock, der die Daten empfängt, die der Ressource zugeordnet sind. Er ist 0, wenn die Ressource nicht existiert.

Hinweis Die Ressource wird erst geladen, wenn die Funktion *LockResource* aufgerufen wird, um das Handle, das von *LoadResource* zurückgeliefert wurde, in einen Zeiger von Typ *Pointer* auf die Ressource-Daten zu übersetzen.

LoadString

Syntax
```
function LoadString(Instance: THandle; ID: Word;
Buffer: PChar; BufferMax: Integer): Integer;
```

Beschreibung Diese Funktion lädt eine String-Ressource, die durch den Parameter *ID* bezeichnet wird, aus der ausführbaren Datei, die dem Modul zugeordnet ist, das durch den Parameter *Instance*

bestimmt wird. Die Funktion kopiert den String in den Puffer, auf den der Parameter *Buffer* zeigt, und fügt ein abschließendes Nullzeichen an.

Instance: Bezeichnet eine Instanz des Moduls, dessen ausführbare Datei die String-Ressource enthält.

ID: Bestimmt den Integerbezeichner des Strings, der geladen werden soll.

Buffer: Zeigt auf den Puffer, der den String aufnehmen soll.

BufferMax: Bestimmt die maximale Anzahl der Zeichen, die in den Puffer kopiert werden sollen. Der String wird abgeschnitten, wenn er länger als die angegebene Zeichenanzahl ist.

Rückgabewert Der Rückgabewert gibt die tatsächlich in den Puffer kopierte Anzahl von Zeichen an. Er ist 0, wenn die String-Ressource nicht existiert.

LoByte

Syntax
```
function LoByte(A: Word): Byte;
```

Beschreibung Diese Inline-Funktion liefert das niederwertige Byte des Integerwertes, der durch den Parameter *A* bestimmt wird.

A: Gibt den Wert an, der umgewandelt werden soll.

Rückgabewert Der Rückgabewert gibt das niederwertige Byte des Wertes an.

LocalAlloc

Syntax
```
function LocalAlloc(Flags, Bytes: Word): THandle;
```

Beschreibung Diese Funktion reserviert die Anzahl von Bytes im lokalen Heap, die vom Parameter *Bytes* angegeben wird. Der Speicherblock kann entweder fest oder verschiebbar sein, wie es im Parameter *Flags* angegeben wird.

Flags: Bestimmt, wie der Speicher reserviert werden soll. Er kann einen oder mehrere der folgenden Werte annehmen:

Wert	Bedeutung
Lmem_Discardable	Reserviert verwerfbaren Speicher. Kann nur mit *Lmem_Moveable* benutzt werden.
Lmem_Fixed	Reserviert festen Speicher.
Lmem_Modify	Modifiziert das Flag *Lmem_Discardable.* Kann nur mit *Lmem_Discardable* benutzt werden.
Lmem_Moveable	Reserviert verschiebbaren Speicher. Kann nicht mit *Lmem_Fixed* benutzt werden.

Wert	Bedeutung
Lmem_NoCompact	Verdichtet oder verwirft keinen Speicher, um die gewünschte Reservierung durchzuführen.
Lmem_NoDiscard	Verwirft keinen Speicher, um die gewünschte Reservierung durchzuführen.
Lmem_ZeroInit	Initialisiert den Speicherinhalt auf 0. Wählen Sie *Lmem_Fixed* oder *Lmem_Moveable*. Dann kombinieren Sie die benötigten anderen Flags durch Verwendung des bitweisen **or**-Operators.

Bytes: Bestimmt die Anzahl von Bytes, die reserviert werden sollen.

Rückgabewert Der Rückgabewert bezeichnet den neu reservierten lokalen Speicherblock, wenn die Funktion erfolgreich war, andernfalls ist er 0.

Hinweis Ist das Datensegment, das den Heap enthält, verschiebbar, so veranlaßt der Aufruf dieser Funktion eine Verschiebung des Segments, wenn Windows den Heap vergrößern muß und die Größe des Heaps in der gegenwärtigen Position nicht vergrößert werden kann.

Eine Anwendung kann Windows daran hindern, das Datensegment zu verlagern, indem sie die Funktion *LockData* aufruft, um das Datensegment zu sperren.

Wurde die Funktion erfolgreich ausgeführt, reserviert sie mindestens die gewünschte Byteanzahl. Der tatsächlich reservierte Speicherplatz kann größer sein. Um den wirklich reservierten Speicher zu bestimmen, rufen Sie die Funktion *LocalSize* auf.

Querverweis *LockData*

LocalCompact

Syntax
```
function LocalCompact(MinFree: Word): Word;
```

Beschreibung Diese Funktion stellt die die vom Parameter *MinFree* angegebene Anzahl an freien Bytes des Speichers zur Verfügung, indem sie den lokalen Heap des Moduls – falls nötig – verdichtet.

Die Funktion überprüft den lokalen Heap auf die angegebene Anzahl an zusammenhängenden freien Bytes. Wird kein Bereich dieser Größe gefunden, verdichtet die Funktion *LocalCompact* den lokalen Speicher, indem sie zuerst alle ungesperrten Blöcke in den oberen Speicher verschiebt.

Falls dadurch der angeforderte Speicher immer noch nicht zur Verfügung gestellt werden kann, verwirft die Funktion nach Möglichkeit solange nicht gesperrte, verschiebbare und

verwerfbare Speicherblöcke, bis der angeforderte Speicher zur Verfügung steht.

MinFree: Bestimmt die Anzahl gewünschter freier Bytes. Wenn *MinFree* 0 ist, liefert die Funktion zwar einen Wert zurück, verdichtet aber den Speicher nicht.

Rückgabewert Der Rückgabewert gibt die Anzahl der Bytes im größten freien Block des lokalen Speichers an.

LocalDiscard

Syntax
```
function LocalDiscard(Mem: THandle): THandle;
```

Beschreibung Diese Funktion verwirft den lokalen Speicherblock, der durch den Parameter *Mem* bestimmt wird.

Der Sperrenzähler des Speicherblocks muß 0 sein. Der lokale Speicherblock wird zwar aus dem Speicher entfernt, aber sein Handle bleibt gültig.

Eine Anwendung kann anschließend das Handle an die Funktion *LocalReAlloc* übergeben, um einen weiteren lokalen Speicherblock zu reservieren, der durch dieses Handle bezeichnet wird.

Mem: Bezeichnet den lokalen Speicherblock, der verworfen werden soll.

Rückgabewert Der Rückgabewert ist 0, wenn die Funktion erfolgreich war, andernfalls ist er identisch mit *Mem*.

LocalFlags

Syntax
```
function LocalFlags(Mem: THandle): Word;
```

Beschreibung Diese Funktion ruft Informationen über den angegebenen lokalen Speicherblock ab.

Mem: Bezeichnet den lokalen Speicherblock.

Rückgabewert Der Rückgabewert enthält eines der folgenden Speicherreservierungs-Flags im oberen Byte.

Wert	Bedeutung
Lmem_Discardable	Block ist als verwerfbar markiert.
Lmem_Discarded	Block wurde verworfen.

Das untere Byte des Rückgabewertes enthält den Referenzzähler des Blocks. Verwenden Sie die Maske *Lmem_LockCount*, um aus dem Rückgabewert den Wert des Sperrenzählers zu ermitteln.

LocalFree

Syntax
```
function LocalFree(Mem: THandle): THandle;
```

Beschreibung Diese Funktion gibt den lokalen Speicherblock frei, der durch den Parameter *Mem* bezeichnet wird, und macht das Handle des Speicherblocks ungültig.

Mem: Bezeichnet den lokalen Speicherblock, der freigegeben werden soll.

Rückgabewert Der Rückgabewert ist 0, wenn die Funktion erfolgreich war, andernfalls ist er identisch mit *Mem*.

LocalHandle

Syntax
```
function LocalHandle(Mem: Word): THandle;
```

Beschreibung Diese Funktion liefert das Handle des lokalen Speicherobjektes, dessen Adresse durch *Mem* bestimmt wird.

Mem: Bestimmt die Adresse eines lokalen Speicherobjektes.

Rückgabewert Der Rückgabewert bezeichnet das lokale Speicherobjekt.

LocalInit

Syntax
```
function LocalInit(Segment, Start, EndPos: Word): Bool;
```

Beschreibung Diese Funktion initialisiert einen lokalen Heap in dem Segment, das durch den Parameter *Segment* angegeben wird.

Segment: Bestimmt die Segmentadresse des Segments, das den lokalen Heap enthalten soll.

Start: Bestimmt die Anfangsadresse des lokalen Heap innerhalb des Segments.

EndPos: Bestimmt die Endadresse des lokalen Heap innerhalb des Segments.

Rückgabewert Der Rückgabewert ist *True*, wenn der Heap initialisiert wurde, andernfalls ist er 0.

Hinweis Ist der Parameter *Start* 0, so gibt der Parameter *EndPos* den Abstand des letzten Bytes des lokalen Heap zu diesem Ausgangspunkt zurück: um zum Beispiel einen Heap von 4096 Byte mit dem ersten Byte bei Byte 0 zu initialisieren, müssen Sie *Start* auf 0 und *EndPos* auf 4095 setzen.

LocalInit ruft die Funktion *GlobalLock* für das Datensegment auf, das den lokalen Heap enthält und stellt so sicher, daß das Datensegment nicht im Speicher verschoben wird.

Der Speicher wird verschoben, wenn folgenden Bedingungen erfüllt sind:

1. Das Datensegment ist verschiebbar.

2. Die Anwendung ruft eine der Funktionen *LocalAlloc* oder *LocalReAlloc* auf, und Windows muß infolgedessen den Heap vergrößern. Kann Windows das Datensegment, das den lokalen Heap enthält, nicht ohne Verschiebung vergrößern, so wird es verschoben.

Eine Anwendung kann Windows ausdrücklich daran hindern, das Datensegment zu verschieben, indem sie die Funktion *LockData* aufruft, um das Datensegment zu sperren.

Eine Anwendung kann diese Sperre entfernen, indem sie die Funktion *UnlockData* aufruft.

LocalLock

Syntax
```
function LocalLock(Mem: THandle): Pointer;
```

Beschreibung Diese Funktion sperrt den lokalen Speicherblock, der durch den Parameter *Mem* bestimmt wird. Der Block wird im Speicher an der gegebenen Adresse gesperrt und sein Referenzzähler wird um 1 erhöht. Gesperrter Speicher kann nicht verschoben oder verworfen werden. Der Block bleibt gesperrt im Speicher, bis sein Referenzzähler durch Verwendung der Funktion *LocalUnlock* auf 0 herabgesetzt wird.

Mem: Bezeichnet den lokalen Speicherblock, der gesperrt werden soll.

Rückgabewert Wenn die Funktion erfolgreich ausgeführt wurde, zeigt der Rückgabewert auf das erste Byte im Speicher des lokalen Blocks, andernfalls ist er 0.

LocalReAlloc

Syntax
```
function LocalReAlloc(Mem: THandle; Bytes, Flags:
Word): THandle;
```

Beschreibung Diese Funktion vergrößert oder verkleinert den lokalen Speicherblock, der durch den Parameter *Mem* bestimmt wird, auf die Anzahl von Bytes, die durch den Parameter *Bytes* vorgegeben ist oder sie ändert die Attribute des angegebenen Speicherblocks.

Mem: Bezeichnet den lokalen Speicherblock, dessen Größe verändert werden soll.

Bytes: Bestimmt die neue Größe des Speicherblocks.

Flags: Bestimmt, wie der lokale Speicherblock verändert werden soll.

Dieser Parameter kann einen oder mehrere der folgenden Werte annehmen:

Wert	Bedeutung
Lmem_Discardable	Speicher ist verwerfbar. Dieses Flag kann nur zusammen mit *Lmem_Modify* benutzt werden.
Lmem_Modify	Speicherflags werden modifiziert. Der Parameter *wBytes* wird ignoriert. Dieses Flag kann nur zusammen mit *Lmem_Discardable* benutzt werden.
Lmem_Moveable	Speicher ist verschiebbar. Wenn *Bytes* 0 ist, verursacht dieses Flag die Freigabe eines vorher festen Blocks, wenn der Referenzzähler des Blocks 0 ist, das Verwerfen eines verschiebbaren Objektes. Wenn *Bytes* ungleich 0 ist und der Block, der von *Mem* bezeichnet wird, fest ist, erlaubt dieses Flag die Verschiebung des veränderten Blocks an eine neue feste Position. Beachten Sie, daß das Handle, das von der Funktion *LocalReAlloc* zurückgeliefert wird, in diesem Fall von dem Handle, *das der Funktion übergeben wurde, verschieden sein kann. Dieses Flag kann nicht mit Lmem_Modifiy* benutzt werden.
Lmem_NoCompact	Speicher wird nicht verdichtet oder verworfen, um die gewünschte Reservierung sicherzustellen. Dieses Flag kann nicht mit *Lmem_Modify* benutzt werden.
Lmem_NoDiscard	Objekte werden nicht verworfen, um die gewünschte Reservierung sicherzustellen. Dieses Flag kann nicht mit *Lmem_Modify* benutzt werden.
Lmem_ZeroInit	Wenn der Block größer wird, werden die zusätzlichen Speicherinhalte mit 0 initialisiert. Dieses Flag kann nicht mit *Lmem_Modify* benutzt werden.

Rückgabewert

Der Rückgabewert bezeichnet den geänderten lokalen Speicher, wenn die Funktion erfolgreich war. Er ist 0, wenn die Größe des lokalen Speicherblocks nicht verändert werden konnte.

Der Rückgabewert ist immer mit dem Parameter *Mem* identisch, wenn nicht das Flag *Lmem_Moveable* benutzt wird, um die Verschiebung eines festen Speicherblocks an eine neue Speicherstelle zu erlauben.

Hinweis Ist das Datensegment, das den Heap enthält, verschiebbar, veranlaßt der Aufruf dieser Funktion eine Verschiebung des Segments, wenn Windows den Heap vergrößern muß und die Größe des Heap in der gegenwärtigen Position nicht vergrössern kann.

Eine Anwendung kann Windows daran hindern, das Datensegment zu verlagern, indem sie die Funktion *LockData* aufruft, um das Datensegment zu sperren.

Querverweis *LockData*

LocalShrink

Syntax
```
function LocalShrink(Seg: THandle; Size: Word): Word;
```

Beschreibung Diese Funktion verkleinert den angegebenen Heap auf die Größe, die vom Parameter *Size* übergeben wird. Die kleinste mögliche Größe des automatischen lokalen Heap ist in der Moduldefinitionsdatei der Anwendung festgelegt.

Seg: Bezeichnet das Segment, das den lokalen Heap enthält.

Size: Bestimmt die Größe des lokalen Heap in Bytes nach der erwünschten Verkleinerung.

Rückgabewert Der Rückgabewert gibt die Größe des lokalen Heap nach der Verkleinerung an.

Hinweis Wenn *Seg* 0 ist, verkleinert die Funktion *LocalShrink* den lokalen Heap im gegenwärtigen Datensegment. Windows wird den Teil des Datensegments nicht verkleinern, der den Stack und die statischen Variablen enthält.

Verwenden Sie die Funktion *GlobalSize*, um die neue Größe des Datensegments zu bestimmen.

Querverweis *GlobalSize*

LocalSize

Syntax
```
function LocalSize(Mem: THandle): Word;
```

Beschreibung Diese Funktion liefert die gegenwärtige Größe des lokalen Speicherblocks, der vom Parameter *Mem* bestimmt wird.

Mem: Bezeichnet den lokalen Speicherblock.

Rückgabewert Der Rückgabewert gibt die Größe (in Bytes) des angegebenen Speicherblocks an. Er ist 0, wenn das gegebene Handle ungültig ist.

Hinweis Die tatsächliche Größe eines Speicherblocks ist manchmal größer, als bei der Reservierung gewünscht wurde.

LocalUnlock

Syntax
```
function LocalUnlock(Mem: THandle): Bool;
```

Beschreibung Diese Funktion macht die Sperrung des durch den Parameter *Mem* bezeichneten lokalen Speicherblocks rückgängig und verringert den Referenzzähler des Blocks um 1.

Der Block ist ungesperrt und kann verschoben und verworfen werden, wenn der Referenzzähler 0 ist.

Mem: Bezeichnet den lokalen Speicherblock, dessen Sperrung aufgehoben werden soll.

Rückgabewert Der Rückgabewert ist *False*, wenn der Referenzzähler des Blocks auf 0 verringert wurde, andernfalls ist er *True*.

LockData

Syntax
```
function LockData(Dummy: Integer): THandle;
```

Beschreibung Diese Funktion sperrt das aktuelle Datensegment im Speicher. Sie ist für Module bestimmt, die verschiebbare Datensegmente haben.

Dummy: Wird nicht benutzt. Er sollte auf 0 gesetzt werden.

Rückgabewert Der Rückgabewert bezeichnet das gesperrte Datensegment, wenn die Funktion erfolgreich war, andernfalls ist er 0.

Querverweis *UnlockData*

LockResource

Syntax
```
function LockResource(ResData: THandle): Pointer;
```

Beschreibung Diese Funktion ermittelt die absolute Speicheradresse der durch den Parameter *ResData* bezeichneten geladenen Ressource. Die Ressource wird im Speicher gesperrt und ihr Referenzzähler wird um 1 erhöht. Die gesperrte Ressource kann nicht verschoben oder verworfen werden. Die Ressource bleibt solange im Speicher gesperrt, bis ihr Referenzzähler durch Aufrufe der Funktion *FreeResource* auf 0 verringert wurde.

Wenn die durch den Parameter *ResData* bezeichnete Ressource verworfen wurde, wird die mit ihr verbundene Ressource-Handle-Funktion (falls vorhanden), aufgerufen, bevor die Funktion *LockResource* zurückkehrt. Die Ressource-Handle-Funktion kann – falls gewünscht – die Ressource erneut berechnen und wieder laden. Nachdem die Ressource-Handle-Funktion zurückgekehrt ist, macht *LockResource* einen weiteren Versuch, die Ressource zu sperren und liefert das Resultat als Rückgabewert.

ResData: Bezeichnet die gewünschte Ressource. Es wird vorausgesetzt, daß dieses Handle durch die Funktion *LoadResource* erzeugt wurde.

Rückgabewert Der Rückgabewert zeigt auf das erste Byte der geladenen Ressource, falls sie gesperrt wurde, andernfalls ist er 0.

Hinweis Die Verwendung des durch die Funktion *FindResource* zurückgelieferten Handles als Parameter *ResData* verursacht einen Fehler. Verwenden Sie die Funktion *UnlockResource*, um die Sperrung einer Ressource rückgängig zu machen, die durch *LockResource* gesperrt wurde.

LockSegment

Syntax
```
function LockSegment(Segment: Word): THandle;
```

Beschreibung Diese Funktion sperrt das Segment, dessen Segmentadresse durch den Parameter *Segment* bestimmt wird. Wenn *Segment* -1 ist, sperrt die Funktion *LockSegment* das aktuelle Datensegment.

Mit Ausnahme von nicht-verwerfbaren Segmenten im Protected Mode (Standard- oder erweiterter 386-Modus) wird das Segment an der gegebenen Adresse im Speicher gesperrt und sein Sperrenzähler um 1 erhöht. Gesperrter Speicher wird nicht verschoben oder verworfen, außer wenn ein Teil des Segments durch die Funktion *GlobalReAlloc* neu belegt wurde. Das Segment bleibt im Speicher gesperrt, bis sein Sperrenzähler auf 0 verringert wird. Im Protected Mode erhöht *LockSegment* ausschließlich den Sperrenzähler von verwerfbaren und automatischen Datensegmenten.

Jedesmal, wenn eine Anwendung *LockSegment* für ein Segment aufruft, muß sie entsprechend *UnlockSegment* für das Segment aufrufen. Die Funktion *UnlockSegment* verringert den Sperrenzähler des Segments.

Andere Funktionen können ebenfalls den Sperrenzähler eines Speicherobjekts beeinflussen. Bei der Beschreibung der Funktion *GlobalFlags* finden sie eine Auflistung dieser Funktionen.

Segment: Bestimmt die Segmentadresse des Segments, das gesperrt werden soll. Wenn *Segment* -1 ist, sperrt die Funktion *LockSegment* das aktuelle Datensegment.

Rückgabewert Der Rückgabewert bezeichnet das Datensegment, wenn die Funktion erfolgreich war.

Wenn das Objekt verworfen wurde oder ein Fehler aufgetreten ist, ist der Rückgabewert 0.

Querverweis *UnlockSegment*

_lopen

Syntax

```
function _lopen(PathName: PChar; ReadWrite: Integer):
Integer;
```

Beschreibung

Diese Funktion öffnet die Datei mit dem durch den Parameter *PathName* festgelegten Namen. Der Parameter *ReadWrite* legt den Zugriffsmodus der Datei fest, wenn die Funktion sie öffnet. Wenn die Datei existiert und nur für Schreibaktionen geöffnet wird, verkürzt die Funktion die Dateilänge auf 0.

Der Dateizeiger wird beim Öffnen auf den Anfang der Datei gesetzt.

PathName: Zeigt auf einen nullterminierten String, der den Namen der zu öffnenden Datei enthält. Der String muß aus Zeichen des ANSI-Zeichensatzes bestehen.

ReadWrite: Bestimmt, ob die Funktion die Datei zum Lesezugriff, zum Schreibzugriff oder zu beidem öffnen soll. Der Parameter *muß* einen der folgenden Werte annehmen:

Wert	Bedeutung
of_Read	Öffnet die Datei nur zum Lesen.
of_ReadWrite	Öffnet die Datei zum Lesen und zum Schreiben.
of_Share_Compat	Öffnet die Datei im Kompatibilitätsmodus und erlaubt das Öffnen der Datei, sooft sie benötigt wird. *OpenFile* mißlingt, wenn die Datei mit einem anderen Modus zur Mehrfachbenutzung geöffnet wurde.
of_Share_Deny_None	Öffnet die Datei, ohne anderen Prozessen Lese- oder Schreibzugriffe auf die Datei zu verweigern. *OpenFile* mißlingt, wenn die Datei durch irgendeinen anderen Prozeß im Kompatibilitätsmodus geöffnet wurde.
of_Share_Deny_Read	Öffnet die Datei und verweigert anderen Prozessen den Lesezugriff auf die Datei. *OpenFile* mißlingt, wenn die Datei im Kompatibilitätsmodus oder zum Lesezugriff durch irgendeinen anderen Prozeß geöffnet wurde.
of_Share_Deny_Write	Öffnet die Datei und verweigert anderen Prozessen den Schreibzugriff auf die Datei. *OpenFile* mißlingt, wenn die Datei im Kompatibilitätsmodus oder zum Schreibzugriff durch irgendeinen anderen Prozeß geöffnet wurde.
of_Share_Exclusive	Öffnet die Datei im Exklusivmodus und verweigert anderen Prozessen sowohl Lese- als auch Schreibzugriffe auf die Datei.

Wert	Bedeutung
	OpenFile hat keinen Erfolg, wenn die Datei in irgendeinem anderen Modus für Lese- oder Schreibzugriffe geöffnet wurde; dies gilt auch für den aktuellen Prozeß.
of_Write	Öffnet die Datei nur zum Schreiben.

Rückgabewert Der Rückgabewert bestimmt das MS-DOS-Datei-Handle, wenn die Funktion die Datei geöffnet hat. Andernfalls ist der Rückgabewert -1.

LoWord

Syntax
```
function LoWord (A: Longint): Word;
```

Beschreibung Diese inline-Funktion liefert das niederwertige Word aus dem 32-Bit-Integer, der durch den Parameter *A* bestimmt wird.

A: Bestimmt den Wert, der untersucht werden soll.

Rückgabewert Der Rückgabewert gibt das niederwertige Word des 32-Bit-Integer an.

LPtoDP

Syntax
```
function LPtoDP(DC: HDC; var Points; Count: Integer):
Bool;
```

Beschreibung Die Funktion *LPtoDP* transformiert die Koordinaten jedes durch den Parameter *Points* angegebenen Punktes vom virtuellen Koordinatensystem der GDI in ein Gerätekoordinatensystem. Die Transformation hängt vom aktuellen Abbildungsmodus ab.

DC: Bezeichnet den Gerätekontext.

Points: Ein Array von Punkten. Jeder Punkt im Array ist eine *TPoint*-Datenstruktur.

Count: Bestimmt die Anzahl von Punkten im Array.

Rückgabewert Der Rückgabewert gibt an, ob alle Punkte umgewandelt sind. Er ist *True*, wenn alle Punkte umgewandelt sind, andernfalls ist er *False*.

_lread

Syntax
```
function _lread(FileHandle: Integer; Buffer: PChar;
Bytes: Integer): Word;
```

Beschreibung Diese Funktion liest Daten aus der durch den Parameter *FileHandle* bestimmten Datei. Der Parameter *Bytes* legt die Anzahl der Bytes fest, die gelesen werden sollen. Die Funktion liefert die tatsächlich gelesene Anzahl von Bytes zurück, wenn die Funktion versucht hat, die Datei am Dateiende zu lesen, ist der Rückgabewert 0.

FileHandle: Bestimmt das DOS-Datei-Handle der Datei, die gelesen werden soll.

Buffer: Zeigt auf einen Puffer, der die Daten aufnehmen soll, die aus der Datei gelesen werden.

Bytes: Bestimmt die Anzahl der Bytes, die aus der Datei gelesen werden sollen.

Rückgabewert Der Rückgabewert liefert die Anzahl der Bytes, die die Funktion tatsächlich aus der Datei gelesen hat, oder -1, wenn die Funktion fehlschlägt.

Der Rückgabewert ist kleiner als *Bytes*, wenn die Funktion das Dateiende erreicht hat, bevor sie die angegebene Anzahl von Bytes gelesen hat.

lstrcat

Syntax
```
function lstrcat(Str1, Str2: PChar): PChar;
```

Beschreibung Diese Funktion hängt *Str2* an den String an, der durch *Str1* bezeichnet wird, schließt den entstehenden String mit einem Nullzeichen ab und liefert einen Zeiger auf den verbundenen String *Str1*.

Alle Strings müssen eine Größe von weniger als 64 KByte haben.

Str1: Zeigt auf ein Zeichen-Array, das einen nullterminierten String enthält, an den die Funktion *Str2* anhängen soll. Dieses Zeichen-Array, das den String enthält, muß groß genug sein, um beide Strings aufzunehmen.

Str2: Zeigt auf einen nullterminierten String, den die Funktion an *Str1* anhängen soll.

Rückgabewert Der Rückgabewert ist bei erfolgreicher Ausführung ein Zeiger auf *Str1*, andernfalls ist er 0.

lstrcmp

Syntax
```
function lstrcmp(Str1, Str2: PChar): Integer;
```

Beschreibung Diese Funktion vergleicht die beiden durch *Str1* und *Str2* bezeichneten Strings lexikographisch und liefert einen Wert zurück, der das Ergebnis des Vergleichs angibt. Der Vergleich

basiert auf der aktuellen Sprache, die vom Anwender im Setup oder in der Systemsteuerung ausgewählt wurde. Der Vergleich unterscheidet zwischen Groß- und Kleinschreibung.

Alle Strings müssen eine Größe von weniger als 64 KByte haben.

Str1: Zeigt auf den ersten nullterminierten String, der verglichen werden soll.

Str2: Zeigt auf den zweiten nullterminierten String, der verglichen werden soll.

Rückgabewert Der Rückgabewert zeigt an, ob *Str1* kleiner, gleich oder größer als *Str2* ist. Dabei gilt:

< 0 *Str1* ist kleiner als *Str2*.
= 0 *Str1* ist mit *Str2* identisch.
> 0 *Str1* ist größer als *Str2*.

lstrcmpi

Syntax
```
function lstrcmpi(Str1, Str2: PChar): Integer;
```

Beschreibung Diese Funktion vergleicht die beiden durch *Str1* und *Str2* bezeichneten Strings lexikographisch und liefert einen Wert zurück, der das Ergebnis des Vergleichs angibt. Der Vergleich basiert auf der aktuellen Sprache, die vom Benutzer im Setup oder in der Systemsteuerung ausgewählt wurde. Zwischen Groß- und Kleinschreibung wird nicht unterschieden.

Alle Strings müssen eine Größe von weniger als 64 KByte haben.

Str1: Zeigt auf den ersten nullterminierten String, der verglichen werden soll.

Str2: Zeigt auf den zweiten nullterminierten String, der verglichen werden soll.

Rückgabewert Der Rückgabewert zeigt an, ob *Str1* kleiner, gleich oder größer als *Str2* ist. Dabei gilt:

< 0 *Str1* ist kleiner als *Str2*.
= 0 *Str1* ist mit *Str2* identisch.
> 0 *Str1* ist größer als *Str2*.

lstrcpy

Syntax
```
function lstrcpy(Str1, Str2: PChar): PChar;
```

Beschreibung Diese Funktion kopiert *Str2* einschließlich des abschließenden Nullzeichens in den Bereich, der von *Str1* bestimmt wird und gibt *Str1* zurück.

Alle Strings müssen eine Größe von weniger als 64 KByte haben.

Str1: Zeigt auf einen Puffer, der den Inhalt von *Str2* aufnehmen soll. Der Puffer muß groß genug für *Str2* sein.

Str2: Zeigt auf den nullterminierten String, der kopiert werden soll.

Rückgabewert Der Rückgabewert ist ein Zeiger auf *Str1*. Er ist 0, wenn die Funktion keinen Erfolg hat.

lstrlen

Syntax
```
function lstrlen(Str: PChar): Integer;
```

Beschreibung Diese Funktion gibt die Länge von *Str* in Bytes ohne das abschließende Nullzeichen zurück.

Der String muß kleiner als 64 KByte sein.

Str: Zeigt auf einen nullterminierten String.

Rückgabewert Es wird die Länge von *Str* zurückgegeben, im Fall eines Fehlers wird kein Wert zurückgeliefert.

_lwrite

Syntax
```
function _lwrite(FileHandle: Integer; Buffer: PChar;
Bytes: Integer): Word;
```

Beschreibung Diese Funktion schreibt Daten in die Datei, die durch den Parameter *FileHandle* bestimmt wird. Der Parameter *Bytes* legt die Anzahl von Bytes fest, die aus dem durch *Buffer* bezeichneten Puffer geschrieben werden sollen.

Der Rückgabewert der Funktion liefert die tatsächlich in die Datei geschriebene Anzahl von Bytes.

FileHandle: Bestimmt das DOS-Datei-Handle der Datei, in die geschrieben werden soll.

Buffer: Zeigt auf einen Puffer, der die Daten enthält, die in die Datei geschrieben werden sollen.

Bytes: Bestimmt die Anzahl von Bytes, die in die Datei geschrieben werden sollen.

Rückgabewert Es wird die Anzahl von Bytes zurückgeliefert, die die Funktion tatsächlich in die Datei geschrieben hat, im Fall eines Fehlers ist der Rückgabewert -1.

Hinweis Der Puffer, der von *Buffer* bestimmt wird, darf nicht über das Ende eines Segments hinausreichen.

Makelong

Syntax
```
function MakeLong(A, B: Word): LongInt;
```

Beschreibung Diese Inline-Funktion erzeugt einen vorzeichenlosen 32-Bit-Integer durch die Verkettung von zwei durch die Parameter *A* und *B* bezeichneten Word-Werten.

A: Bestimmt das niederwertige Word des neuen LongInt.

B: Bestimmt das höherwertige Word des neuen LongInt.

Rückgabewert Der Rückgabewert ist der neue vorzeichenlose LongInt.

MakeProcInstance

Syntax
```
function MakeProcInstance(Proc: TFarProc; Instance:
THandle): TFarProc;
```

Beschreibung Diese Funktion erzeugt die Adresse einer Prozedurinstanz.

Eine Prozedurinstanzadresse zeigt auf den Prologcode, der vor der Ausführung der Funktion ausgeführt wird. Der Prolog verknüpft das Datensegment der Instanz, die durch den Parameter *Instance* bezeichnet wird, mit der Funktion, auf die der Parameter *Proc* zeigt. Wenn die Funktion ausgeführt wird, hat sie Zugriff auf Variablen und Daten im Datensegment dieser Instanz.

Proc: Ist die Adresse einer Prozedurinstanz.

Instance: Bezeichnet die Instanz, die dem gewünschten Datensegment zugeordnet ist.

Rückgabewert Der Rückgabewert zeigt bei erfolgreicher Ausführung auf die Funktion, andernfalls ist er 0.

Hinweis Die Funktion *MakeProcInstance* darf nur verwendet werden, um auf Funktionen von Instanzen des aktuellen Moduls zuzugreifen. Diese Funktion ist für Bibliotheksmodule nicht erforderlich.

Nachdem *MakeProcInstance* für eine bestimmte Funktion aufgerufen worden ist, sollten alle Aufrufe dieser Funktion über die abgerufene Adresse erfolgen.

MakeProcInstance erzeugt mehr als eine Prozedurinstanz. Eine Anwendung sollte *MakeProcInstance* nicht öfter als einmal mit derselben Funktion und demselben Instanz-Handle benutzen, um Vergeudung von Speicherplatz zu vermeiden.

Um ein Datensegment mit einer Funktion zu verknüpfen, muß die Funktion durch eine EXPORTS-Anweisung in der Moduldefinitionsdatei exportiert werden.

MapDialogRect

Syntax `procedure MapDialogRect(Dlg: HWnd; var Rect: TRect;)`

Beschreibung Diese Prozedur konvertiert die in der *TRect*-Struktur angegebenen Dialogfenstereinheiten in Bildschirmeinheiten. Dialogfenstereinheiten sind in Ausdrücken der aktuellen Dialog-Basiseinheit festgelegt, die sich aus der durchschnittlichen Höhe und Breite der Zeichen in der Systemschrift ergibt. Eine horizontale Einheit ist ein Viertel der Dialog-Basisbreiteneinheit, eine vertikale Einheit ein Achtel der Dialog-Basishöheneinheit.

Die Funktion *GetDialogBaseUnits* gibt die Dialog-Basiseinheiten in Pixel zurück. *MapDialogRect* ersetzt die Dialogfenstereinheiten in *Rect* durch Bildschirmeinheiten (Pixel), so daß das Rechteck benutzt werden kann, um ein Dialogfenster zu erstellen oder ein Steuerelement innerhalb eines Fensters zu positionieren.

Dlg: Bezeichnet ein Dialogfenster.

Rect: Eine *TRect*-Datenstruktur, die die zu konvertierenden Dialogfensterkoordinaten enthält.

Hinweis Das Handle in *Dlg* muß mit einer der Funktionen *CreateDialog* oder *DialogBox* erzeugt werden.

Querverweis *GetDialogBaseUnits*

MapVirtualKey

Syntax `function MapVirtualKey(Code, MapType: Word): Word;`

Beschreibung Diese Funktion nimmt einen virtuellen Tastencode oder einen Scan-Code für eine Taste entgegen und gibt den zugehörigen Scan-Code, virtuellen Tastencode oder ASCII-Wert zurück. Der Wert des Parameters *MapType* legt den Typ der Konversion fest, den diese Funktion ausführt.

Code: Bestimmt den virtuellen Tastencode oder Scan-Code für eine Taste. Die Interpretation des Parameters *Code* hängt vom Wert des Parameters *MapType* ab.

MapType: Bestimmt den Typ der Konversion, die ausgeführt werden soll. Der Parameter *MapType* kann einen der folgenden Werte annehmen:

Wert	Bedeutung
0	Der Parameter *Code* bestimmt einen virtuellen Tastencode, die Funktion gibt den zugehörigen Scan-Code zurück.

Wert	Bedeutung
1	Der Parameter *Code* bestimmt einen Scan-Code, die Funktion gibt den zugehörigen virtuellen Tastencode zurück.
2	Der Parameter *Code* bestimmt einen virtuellen Tastencode, die Funktion gibt den zugehörigen ASCII-Wert zurück, der gilt, wenn die Shift-Taste nicht gedrückt ist. Andere Werte sind reserviert.

Rückgabewert Der Rückgabewert hängt vom Wert der Parameter *Code* und *MapType* ab.

MessageBeep

Syntax
```
procedure MessageBeep(BeepType: Word);
```

Beschreibung Diese Prozedur erzeugt einen Warnton am Systemlautsprecher.

BeepType: Wird nicht verwendet und sollte auf 0 gesetzt werden.

MessageBox

Syntax
```
function MessageBox(WndParent: HWnd; Txt, Caption:
PChar; TextType: Word): Integer;
```

Beschreibung Diese Funktion erzeugt ein Fenster und stellt es mit einer von der Anwendung bereitgestellten Meldung und Kopfzeile dar. Hinzu kommt eine Kombination der vordefinierten Symbole und Schalter, die im Hinweis beschrieben sind.

WndParent: Bezeichnet das Fenster, dem das Meldungsfenster gehört.

Txt: Zeigt auf einen nullterminierten String, der die anzuzeigende Meldung enthält

Caption: Zeigt auf einen nullterminierten String, der als Überschrift des Dialogfensters verwendet wird. Wenn der Parameter *Caption* 0 ist, wird die Standardüberschrift benutzt.

TextType: Bestimmt den Inhalt des Dialogfensters. Er kann jede Kombination der im Hinweis beschriebenen Werte sein, verknüpft durch den bitweisen **or**-Operator.

Rückgabewert Wenn nicht genug Speicherplatz vorhanden ist, um das Dialogfenster zu erzeugen, wird 0 zurückgegeben. Andernfalls ist der Rückgabewert einer der folgenden Menüoptionswerte des Dialogfensters:

Wert	Bedeutung
idAbort	Aktionsschalter *Beenden* gedrückt.
idCancel	Aktionsschalter *Abbrechen* gedrückt.
idIgnore	Aktionsschalter *Ignorieren* gedrückt.
idNo	Aktionsschalter *Nein* gedrückt.
idOk	Aktionsschalter *OK* gedrückt.
idRetry	Aktionsschalter *Wiederholen* gedrückt.
idYes	Aktionsschalter *Ja* gedrückt.

Wenn das Meldungsfenster über einen Aktionsschalter *Abbrechen* verfügt, wird der Wert *IdCancel* zurückgegeben, wenn entweder die Taste Esc oder der Aktionsschalter *Abbrechen* gedrückt wird. Wenn das Meldungsfenster über keinen Aktionsschalter *Abbrechen* verfügt, hat das Drücken der Taste Esc keine Wirkung.

Hinweis

Wenn ein systemmodales Meldungsfenster erzeugt wird, um anzuzeigen, daß im System der Speicherplatz knapp wird, sollten die Strings, die als die Parameter *Txt* und *Caption* übergeben werden, nicht einer Ressourcendatei entnommen werden, da ein Versuch, die Ressource zu laden, fehlschlagen könnte.

Wenn eine Anwendung die Funktion *MessageBox* aufruft und die Flags *mb_IconHand* und *mb_SystemModal* für den Parameter *Type* vorgibt, stellt Windows das resultierende Meldungsfenster ungeachtet des verfügbaren Speicherplatzes dar. Wenn diese Flags gesetzt sind, wird die Länge des Meldungsfenstertextes auf eine Zeile begrenzt.

Wenn ein Meldungsfenster erzeugt wird, während ein Dialogfenster präsent ist, sollte man das Handle des Dialogfensters als Parameter *WndParent* benutzen. Dieser Parameter sollte kein untergeordnetes Fenster, wie etwa das Steuerelement eines Dialogfensters, bezeichnen.

Wert	Bedeutung
mb_AbortRetryIgnore	Meldungsfenster enthält drei Schalter: *Beenden*, *Wiederholen* und *Ignorieren.*
mb_ApplModal	Der Anwender muß auf das Meldungsfenster reagieren, bevor er die Arbeit in dem durch den Parameter *WndParent* bezeichneten Fenster fortsetzt Der Anwender kann sich jedoch zu den Fenstern anderer Anwendungen bewegen und in diesen Fenstern arbeiten. *mb_ApplModal* ist der Standardwert, wenn weder *mb_SystemModal* noch *mb_TaskModal* angegeben ist.

Wert	Bedeutung
mb_DefButton1	Erster Schalter ist die Voreinstellung. Man beachte, daß der erste Schalter immer die Voreinstellung ist, wenn nicht *mb_DefButton2* oder *mb_DefButton3* angegeben ist.
mb_DefButton2	Zweiter Schalter ist Voreinstellung.
mb_DefButton3	Dritter Schalter ist Voreinstellung.
mb_IconAsterisk	Gleichbedeutend mit *mb_IconInformation.*
mb_IconExclamation	Ein Ausrufezeichensymbol erscheint im Meldungsfenster.
mb_IconHand	Gleichbedeutend mit *mb_Iconstop*.
mb_IconInformation	Ein Symbol, bestehend aus einem kleinen *i* in einem Kreis, erscheint im Meldungsfenster.
mb_IconQuestion	Ein Fragezeichensymbol erscheint im Meldungsfenster.
mb_IconStop	Ein Stopzeichensymbol erscheint im Meldungsfenster.
mb_Ok	Meldungsfenster enthält einen Schalter: *Ok*.
mb_OkCancel	Meldungsfenster enthält zwei Schalter: *Ok* und *Abbrechen*.
mb_RetryCancel	Meldungsfenster enthält zwei Schalter: *Abbrechen* und *Wiederholen*.
mb_SystemModal	Alle Anwendungen werden unterbrochen, bis der Anwender dem Meldungsfenster antwortet. Wenn die Anwendung nicht *mb_IconHand* vorgibt, wird das Meldungsfenster erst modal, nachdem es erstellt worden ist; folglich empfangen das übergeordnete und andere Fenster weiterhin Botschaften, die sich aus seiner Aktivierung ergeben. Systemmodale Meldungsfenster werden dazu benutzt, den Benutzer von ernsten, möglicherweise schadensverursachenden Fehlern zu benachrichtigen, die sofortige Aufmerksamkeit erfordern (beispielsweise Speicherüberlauf).
mb_TaskModal	Gleichbedeutend mit *mb_ApplModal* mit der Ausnahme, daß alle Hauptfenster, die zur aktuellen Task gehören, gesperrt werden, wenn der Parameter *WndParent* 0 ist. Dieses Flag sollte benutzt werden, wenn die aufrufende Anwendung oder Bibliothek kein verfügbares Fenster-Handle hat, aber eines benötigt, um Eingaben an andere Fenster der aktuellen Anwendung zu vermeiden, ohne andere Anwendungen zu unterbrechen.

Wert	Bedeutung
mb_YesNo	Meldungsfenster enthält zwei Schalter: *Ja* und *Nein*.
mb_YesNoCancel	Meldungsfenster enthält drei Schalter: *Ja*, *Nein* und *Abbrechen*.

ModifyMenu

Syntax

```
function ModifyMenu(Menu: HMenu; Position, Flags,
IDNewItem: Word; NewItem: PChar): Bool;
```

Beschreibung

Diese Funktion ändert eine vorhandene Menüoption an der Position, die durch den Parameter *Position* bestimmt wird. Die Anwendung gibt den neuen Status der Menüoption vor, indem sie die Werte im Parameter *Flags* einstellt.

Wenn diese Funktion ein der Menüoption zugeordnetes Pop-Up-Menü ersetzt, entfernt sie das alte Pop-Up-Menü und gibt den Speicher frei, der vom Pop-Up-Menü benutzt wurde.

Menu: Bezeichnet das Menü, das verändert werden soll.

Position: Bestimmt die zu verändernde Menüoption. Die Interpretation des Parameters *Position* hängt von der Einstellung des Parameters *Flags* ab.

Flags	Position:
mf_ByPosition	Bestimmt die Position einer existierenden Menüoption. Die erste Option im Menü ist an Position 0.
mf_ByCommand	Bestimmt die Befehls-ID der vorhandenen Menüoption.

Flags: Bestimmt, wie der Parameter *Position* interpretiert wird und enthält Informationen über die Veränderungen, die an der Menüoption vorgenommen werden sollen. Er besteht aus einem oder mehreren Werten, die im folgenden Abschnitt Hinweis aufgeführt sind.

IDNewItem: Gibt entweder die Befehls-ID der modifizierten Menüoption an oder, wenn *Flags* auf *mf_Popup* gesetzt ist, das Menü-Handle des Pop-Up-Menüs.

NewItem: Bestimmt den Inhalt der geänderten Menüoption. Wenn *Flags* auf *mf_String* gesetzt ist (Voreinstellung), ist *NewItem* ein Zeiger auf einen nullterminierten String. Ist *Flags* stattdessen auf *mf_Bitmap* gesetzt, enthält *NewItem* in seinem niederwertigen Word ein Bitmap-Handle (*HBitmap*). Wenn *Flags* auf *mf_OwnerDraw* gesetzt ist, bestimmt *NewItem* einen

von der Anwendung bereitgestellten 32-Bit-Wert, den die Anwendung benutzen kann, um zusätzliche Daten zu verwalten, die mit der Menüoption verbunden sind. Dieser 32-Bit-Wert ist für die Anwendung im Feld *itemData* der Struktur verfügbar, auf die der Parameter *Param* der Botschaften *wm_MeasureItem* und *wm_DrawItem* zeigt.

Diese Botschaften werden gesendet, wenn die Menüoption erstmals angezeigt oder verändert wird.

Rückgabewert Der Rückgabewert ist *True*, wenn die Funktion erfolgreich war, andernfalls ist er *False*.

Hinweis Immer, wenn sich ein Menü ändert (unabhängig davon, ob sich das Menü in einem sichtbaren Fenster befindet), sollte die Anwendung *DrawMenuBar* aufrufen. Die Veränderung der Attribute vorhandener Menüoptionen läßt sich am schnellsten mit den Funktionen *CheckMenuItem* und *EnableMenuItem* bewerkstelligen.

Jede der folgenden Gruppen enthält Flags, die nicht zusammen benutzt werden sollten:

mf_ByCommand und *mf_ByPosition*
mf_Disabled, mf_Enabled und *mf_Grayed*
mf_Bitmap, mf_String, mf_OwnerDisplay und *mf_Separator*
mf_MenuBarBreak und *mf_MenuBreak*
mf_Checked und *mf_Unchecked*

Die folgende Liste beschreibt die Flags, die im Parameter *Flags* gesetzt werden können:

Wert	Bedeutung
mf_Bitmap	Benutzt ein Bitmap als Option. Das niederwertige Word des Parameters *NewItem* enthält das Handle des Bitmaps.
mf_ByCommand	Bestimmt, daß der Parameter *Position* die Nummer der Steuerelement-ID der Menüoption angibt (Voreinstellung).
mf_ByPosition	Bestimmt, daß der Parameter *Position* statt einer ID-Nummer eine zu verändernde Menüoption bezeichnet.
mf_Checked	Plaziert eine Auswahlmarkierung neben der Menüoption. Wenn die Anwendung Auswahlmarkierung-Bitmaps zur Verfügung stellt (siehe auch Funktion *SetMenuItemBitmaps*), bewirkt das Setzen dieses Flags die Darstellung des Bitmaps neben der Menüoption.
mf_Disabled	Sperrt die Menüoption, so daß sie nicht ausgewählt werden kann, versetzt sie jedoch nicht in den nichtaktivierbaren Zustand.

Wert	Bedeutung
mf_Enabled	Aktiviert die Menüoption (sie kann ausgewählt werden) und hebt die Nichtaktivierbarkeit auf.
mf_Grayed	Sperrt die Menüoption, so daß sie nicht ausgewählt werden kann, und versetzt sie in den nichtaktivierbaren Zustand.
mf_MenuBarBreak	Gleichbedeutend mit *mf_MenuBreak* mit der Ausnahme, daß bei Pop-up-Menüs die neue Spalte von der alten Spalte durch eine vertikale Linie getrennt wird.
mf_MenuBreak	Plaziert die Menüoption für statische Menüleistenelemente in einer neuen Zeile. Bei Pop-up--Menüs wird die Menüoption in einer neuen Spalte ohne Trennlinie zwischen den Spalten plaiert.
mf_OwnerDraw	Gibt an, daß die Option eine von der Anwendung gezeichnete Option ist. Das Fenster, dem das Menü gehört, empfängt eine Botschaft *wm_MeasureItem,* wenn das Menü zum ersten Mal dargestellt wird, um Höhe und Breite der Menüoption abzurufen. Die Botschaft *wm_DrawItem* wird immer dann an die Anwendung gesendet, wenn die Anwendung das Erscheinungsbild der Menüoption aktualisieren muß. Diese Opion ist für Hauptmenüoptionen nicht gültig.
mf_Popup	Bestimmt, daß der Menüoption ein Pop-up-Menü zugeordnet ist. Der Parameter *IDNewItem* bezeichnet ein Handle auf ein Pop-up-Menü, das der Option zugeordnet werden soll. Man sollte das Flag *mf_OwnerDraw* benutzen, um ein Haupt-Pop-up-Menü oder ein hierarchisches Pop-up-Menü an eine Pop-up-Menüoption anzufügen.
mf_Separator	Zeichnet eine horizontale Trennlinie. Man kann dieses Flag in einem Pop-up-Menü benutzen. Diese Linie kann nicht gesperrt, hervorgehoben oder in nichtaktivierbaren Zustand versetzt werden. Windows ignoriert die Parameter *NewItem* und *IDNewItem*.
mf_String	Bestimmt, daß die Menüoption ein String ist. Der Parameter *NewItem* zeigt auf den String für die Menüoption.
mf_Unchecked	Plaziert keine Auswahlmarkierung neben der Option (Voreinstellung). Wenn die Anwendung Auswahlmarkierung-Bitmaps zur Verfügung stellt (siehe auch *SetMenuItemBitmaps*), bewirkt das Setzen dieses Flags die Darstellung des Bitmaps neben der Menüoption.

MoveTo

Syntax
```
function MoveTo(DC: HDC; X, Y: Integer): LongInt;
```

Beschreibung Diese Funktion stellt den Punkt, der durch die Parameter *X* und *Y* bestimmt wird, als aktuelle Position ein.

DC: Bezeichnet den Gerätekontext.

X: Bestimmt die virtuelle x-Koordinate der neuen Position.

Y: Bestimmt die virtuelle y-Koordinate der neuen Position.

Rückgabewert Es werden die x- und y-Koordinate der vorherigen Position zurückgeliefert. Die y-Koordinate liegt im höherwertigen, die x-Koordinate im niederwertigen Word.

Hinweis Obwohl *MoveTo* keine Ausgabe bewirkt, beeinflußt sie andere Ausgabefunktionen, die die aktuelle Position benutzen.

MoveWindow

Syntax
```
procedure MoveWindow(Wnd: HWnd; X, Y, Width, Height:
Integer; Repaint: Bool);
```

Beschreibung Diese Prozedur bewirkt, daß eine Botschaft *wm_Size* an das vorgegebene Fenster gesendet wird. Die Parameter *X*, *Y*, *Width* und *Height* geben die neue Größe des Fensters vor.

Wnd: Bezeichnet ein untergeordnetes oder Pop-up-Fenster.

X: Bestimmt die neue x-Koordinate der oberen linken Ecke des Fensters.

Y: Bestimmt die neue y-Koordinate der oberen linken Ecke des Fensters. Für Pop-up-Fenster sind *X* und *Y* in Bildschirmkoordinaten gegeben (relativ zur oberen linken Ecke des Bildschirms). Für untergeordnete Fenster gelten sie in Client-Koordinaten (relativ zur oberen linken Ecke des Client-Bereichs des übergeordneten Fensters).

Width: Bestimmt die neue Breite des Fensters.

Height: Bestimmt die neue Höhe des Fensters.

Repaint: Bestimmt, ob das Fenster nach der Verschiebung neu gezeichnet werden soll. Wenn *Repaint* 0 ist, wird das Fenster nicht neu gezeichnet.

Hinweis Jedes untergeordnete oder Pop-up-Fenster hat eine minimale Höhe und Breite. Diese Werte hängen von Art und Inhalt des Fensters ab. Jeder Versuch, die Höhe und Breite mit der Prozedur *MoveWindow* kleiner zu machen, schlägt fehl. Die Botschaft *wm_Size*, die von dieser Funktion erzeugt wird, gibt die

neue Breite und Höhe des Client-Bereichs des Fensters weiter, nicht die des gesamten Fensters.

MulDiv

Syntax
```
function MulDiv(Number, Numerator, Denominator: Integer): Integer;
```

Beschreibung Diese Funktion multipliziert zwei Integerwerte und dividiert das Ergebnis durch einen dritten. Der Rückgabewert ist das Endergebnis, gerundet auf den nächstliegenden Integer.

Number: Bestimmt die Zahl, die mit *Numerator* multipliziert werden soll.

Numerator: Bestimmt die Zahl, die mit *Number* multipliziert werden soll.

Denominator: Bestimmt die Zahl, durch die das Resultat der Multiplikation dividiert werden soll.

Rückgabewert Bei erfolgreicher Ausführung wird das Ergebnis der Multiplikation und der Division zurückgeliefert. Wenn entweder ein Überlauf auftritt oder *Denominator* auf 0 gesetzt war, ist der Rückgabewert 32767 oder -32767.

NetBIOSCall

Syntax
```
procedure NetBIOSCall;
```

Beschreibung Diese Prozedur ermöglicht es einer Anwendung, den NETBIOS-Interrupt 5CH aufzurufen. Eine Anwendung sollte *NetBIOSCall* aufrufen, anstatt einen NETBIOS-Interrupt 5CH direkt einzuleiten, um die Kompatibilität mit zukünftigen Microsoft-Produkten sicherzustellen.

Diese Prozedur kann nur innerhalb eines Assemblerteils aufgerufen wrden, da die Register vor dem Aufruf korrekt besetzt sein müssen. Man kann dazu eine als **external** deklarierte Routine oder den integrierten Assembler verwenden.

Bevor *NetBIOSCall* aufgerufen wird, müssen alle Register wie für einen tatsächlichen INT 5CH gesetzt werden. Bei Ende der Funktion sind alle Register die gleichen, wie nach der entsprechenden Funktion INT 5CH. Ein Aufruf könnte damit wie folgt aussehen:

```
asm
   MOV    AX,xxxx
   .
   .
   Dos3Call
end ;
```

OemKeyScan

Syntax
```
function OemKeyScan(OemChar: Word): LongInt;
```

Beschreibung Diese Funktion überträgt die OEM-ASCII-Codes 0 bis $0FF in die OEM Scan-Codes und Shift-Zustände. Sie stellt dadurch Informationen bereit, die es einem Programm ermöglichen, einen OEM-Text an ein anderes Programm zu senden, indem es eine Tastatureingabe simuliert. Speziell hierfür wird sie von Windows insbesondere im erweiterten 386-Modus verwendet.

OemChar: Betimmt den ASCII-Wert des OEM-Zeichens.

Rückgabewert Der Rückgabewert enthält in seinem niederwertigen Word den Scan-Code des OEM-Zeichens, das durch den Parameter *OemChar* bezeichnet wird. Das höherwertige Word des Rückgabewerts enthält Flags, die den Shift-Zustand angeben. Die einzelnen Flags haben folgende Bedeutung:

Bit	Bedeutung
2	Ctrl -Taste ist gedrückt.
1	Eine Shift -Taste ist gedrückt.

Wenn das Zeichen nicht in der OEM-Zeichentabelle definiert ist, enthält sowohl das höherwertige als auch das niederwertige Word des Rückgabewerts -1.

Hinweis Diese Funktion sieht keine Übersetzungen für Zeichen vor, die Ctrl -, Alt - oder Sondertasten benötigen. Zeichen, die von ihr nicht übersetzt werden, müssen durch Simulation der Eingabe mittels des ALT+Nummernblock-Mechanismus kopiert werden. Die Taste Numlock darf dabei nicht arretiert sein.

Diese Funktion ruft in neueren Versionen des Tastaturtreibers die Funktion *VkKeyScan* auf.

OemToAnsi

Syntax
```
function OemToAnsi(OemStr, AnsiStr: PChar): Bool;
```

Beschreibung Diese Funktion übersetzt den String, auf den der Parameter *OemStr* zeigt, vom OEM-Zeichensatz in den ANSI-Zeichensatz. Der String kann größer als 64 KByte sein.

OemStr: Zeigt auf einen nullterminierten String aus dem OEM-Zeichensatz.

AnsiStr: Zeigt auf die Position, an die der übersetzte String kopiert werden soll. *AnsiStr* kann mit *OemStr* identisch sein, um den String an der aktuellen Position zu übersetzen.

Rückgabewert Der Rückgabewert ist immer -1.

OemToAnsiBuff

Syntax
```
procedure OemToAnsiBuff(OemStr, AnsiStr: PChar; Length: Integer);
```

Beschreibung Diese Prozedur übersetzt den String in dem Puffer, auf den der Parameter *OemStr* zeigt, vom OEM- in den ANSI-Zeichensatz.

OemStr: Zeigt auf einen Puffer, der ein oder mehrere Zeichen aus dem OEM-Zeichensatz enthält.

AnsiStr: Zeigt auf die Position, an die der String kopiert werden soll. Der Parameter *AnsiStr* kann mit *OemStr* identisch sein, um den String an der aktuellen Position zu übersetzen.

Length: Bestimmt die Zahl der Zeichen in dem Puffer, der durch den Parameter *OemStr* bezeichnet ist. Falls *Length* 0 ist, ist die Länge 64 KByte (65536).

OffsetClipRgn

Syntax
```
function OffsetClipRgn(DC: HDC; X, Y: Integer): Integer;
```

Beschreibung Diese Funktion verschiebt die Clipping-Region des gegebenen Gerätes um die vorgegebenen Offsetwerte. Sie verschiebt die Region um *X* Einheiten entlang der x-Achse und um *Y* Einheiten entlang der y-Achse.

DC: Bezeichnet den Gerätekontext.

X: Bezeichnet die Anzahl der virtuellen Einheiten, um die nach links oder rechts verschoben werden soll.

Y: Bezeichnet die Zahl der virtuellen Einheiten, um die nach oben oder unten verschoben werden soll.

Rückgabewert Der Rückgabewert bezeichnet den Typ der neuen Region. Er kann einen der folgenden Werte annehmen:

Wert	Bedeutung
ComplexRegion	Die Clipping-Region hat überlappende Grenzen.
Error	Gerätekontext ist ungültig.
NullRegion	Clipping-Region ist leer.
SimpleRegion	Clipping-Region hat keine überlappenden Grenzen.

OffsetRect

Syntax
```
procedure OffsetRect(var Rect: TRect; X, Y: Integer);
```

Beschreibung Diese Prozedur verschiebt das gegebene Rechteck um *X* Einheiten entlang der x-Achse und um *Y* Einheiten entlang der Y-Achse. Die Parameter *X* und *Y* sind vorzeichenbehaftet, so

daß das Rechteck nach rechts oder links und nach oben oder unten bewegt werden kann.

Rect: Eine *TRect*-Datenstruktur, die das zu verschiebende Rechteck enthält.

X: Gibt den Betrag der Verschiebung nach rechts oder links an. Zum Verschieben nach links muß er negativ sein.

Y: Gibt den Betrag der Verschiebung nach oben oder unten an. Zum Verschieben nach oben muß er negativ sein.

Hinweis Die Koordinatenwerte eines Rechtecks dürfen nicht größer als 32767 oder kleiner als -32768 sein. Die Parameter *X* und *Y* müssen sorgfältig gewählt werden, um ungültige Rechtecke zu vermeiden.

OffsetRgn

Syntax
```
function OffsetRgn(Rgn: HRgn; X, Y: Integer): Integer;
```

Beschreibung Diese Funktion verschiebt die angegebene Region um die gegebenen Offsetwerte und zwar um *X* Einheiten entlang der x-Achse und um *Y* Einheiten entlang der y-Achse.

Rgn: Bezeichnet die zu verschiebenden Region.

X: Bezeichnet die Anzahl der Einheiten, um die nach links oder rechts verschoben werden soll.

Y: Bezeichnet die Anzahl der Einheiten, um die nach oben oder unten verschoben werden soll.

Rückgabewert Der Rückgabewert bezeichnet den Typ der neuen Region. Er kann einen der folgenden Werte annehmen:

Wert	Bedeutung
ComplexRegion	Region hat überlappende Grenzen.
Error	Handle der Region ist ungültig.
NullRegion	Region ist leer.
SimpleRegion	Region hat keine überlappenden Grenzen.

Hinwweis Die Koordinatenwerte der Region dürfen nicht größer als 32767 und nicht kleiner als -32768 sein. Die Parameter *X* und *Y* müssen sorgfältig gewählt werden, um ungültige Regionen zu vermeiden.

OffsetViewportOrg

Syntax
```
function OffsetViewportOrg(DC: HDC; X, Y: Integer):
LongInt;
```

Beschreibung Diese Funktion verschiebt den Ursprung des Grafikfensters relativ zu den aktuellen Koordinaten gemäß folgender Formel:

xNeuVO = xAltVO + *X* ; yNeuVO = yAltVO + *Y*

Der neue Ursprung ist die Summe des aktuellen Ursprungs und der Werte *X* und *Y*.

DC: Bezeichnet den Gerätekontext.

X: Bestimmt die Anzahl der gerätespezifischen Einheiten, die zur x-Koordinate des aktuellen Ursprungs addiert werden sollen.

Y: Bestimmt die Anzahl der gerätespezifischen Einheiten, die zur y-Koordinate des aktuellen Ursprungs addiert werden sollen.

Rückgabewert Dies ist der vorherige Ursprung des Fensters in gerätespezifischen Koordinaten. Die vorherige y-Koordinate liegt im höherwertigen, die vorherige x-Koordinate im niederwertigen Word.

OffsetWindowOrg

Syntax
```
function OffsetWindowOrg(DC: HDC; X, Y: Integer):
LongInt;
```

Beschreibung Diese Funktion verändert den Ursprung des Fensters relativ zu den aktuellen Werten gemäß der folgenden Formel:

xNeuWO = xAltWO + *X*; yNeuWO = yAltWO + *Y*

Der neue Ursprung ist die Summe des aktuellen Ursprungs und der Werte *X* und *Y*.

DC: Bezeichnet den Gerätekontext.

X: Bestimmt die Anzahl der virtuellen Einheiten, die zur x-Koordinate des aktuellen Ursprungs addiert werden sollen.

Y: Bezeichnet die Anzahl der virtuellen Einheiten, die zur y-Koordinate des aktuellen Ursprungs addiert werden sollen.

Rückgabewert Dies ist der vorherige Fensterursprung in virtuellen Koordinaten. Die vorherige y-Koordinate steht im höherwertigen, die vorherige x-Koordinate im niederwertigen Word.

OpenClipboard

Syntax
```
function OpenClipboard(Wnd: HWnd): Bool;
```

Beschreibung Diese Funktion öffnet die Zwischenablage zur Untersuchung und hindert andere Anwendungen daran, den Inhalt der Zwischenablage zu ändern.

Wnd: Bezeichnet das Fenster, das mit der offenen Zwischenablage verbunden werden soll.

Rückgabewert Der Rückgabewert bezeichnet den Status der Zwischenablage. Er ist *True*, wenn die Zwischenablage geöffnet ist und *False*, wenn die Zwischenablage schon von einer anderen Anwendung geöffnet wurde.

Hinweis Eine Anwendung sollte die Funktion *CloseClipboard* nach jedem erfolgreichen Aufruf von *OpenClipboard* verwenden.

Querverweis *CloseClipboard*

OpenComm

Syntax
```
function OpenComm(ComName: PChar; InQueue, OutQueue:
Word): Integer;
```

Beschreibung Diese Funktion öffnet eine Schnittstelle und weist ihr ein *Cid*-Handle zu. Die Schnittstelle wird für eine Standardkonfiguration initialisiert. Um sie mit anderen Werten zu initialisieren, sollte die Funktion *SetCommState* benutzt werden. Die Funktion *OpenComm* belegt Speicherplatz für Empfangs- und Sende-Warteschlangen. Die Warteschlangen werden von der interruptgesteuerten Sende-/Empfangssoftware benutzt.

ComName: Weist auf einen String, der COMs oder LPTs enthält, wobei *n* im Bereich von 1 bis zur Anzahl der verfügbaren Schnittstellen für den jeweiligen I/O-Anschluß liegt.

InQueue: Bestimmt die Größe der Empfangs-Warteschlange.

OutQueue: Bestimmt die Größe der Sende-Warteschlange.

Rückgabewert Der Rückgabewert bezeichnet die geöffnete Schnittstelle. Im Fall eines Fehlers wird einer der folgenden negativen Fehlerwerte zurückgeliefert:

Wert	Bedeutung
ie_BadID	Ungültige oder nicht unterstützte ID.
ie_BaudRate	Nicht unterstützte Baudrate.
ie_ByteSize	Ungültige Bytegröße.
ie_Default	Fehler in Standardparametern.
ie_Hardware	Hardware nicht vorhanden.
ie_Memory	Es können keine Warteschlangen eingerichtet werden.
ie_NOpen	Gerät nicht bereit.
ie_Open	Gerät bereits bereit.

Hinweis LPT-Anschlüsse sind nicht interruptgesteuert – für diese werden die Parameter *InQueue* und *OutQueue* ignoriert und die Größe der Warteschlange wird auf 0 gesetzt.

OpenFile

Syntax

```
function OpenFile(FileName: PChar; var ReOpenBuff:
TOFStruct; Style: Word): Integer;
```

Beschreibung

Diese Funktion legt eine Datei an, öffnet eine Datei, öffnet sie wieder oder löscht eine Datei.

FileName: Zeigt auf einen nullterminierten String, der die zu öffnende Datei benennt. Der String muß aus Zeichen aus dem ANSI-Zeichensatz bestehen.

ReOpenBuff: TOFStruct-Datenstruktur, die dazu bestimmt ist, Informationen über die Datei aufzunehmen, wenn diese erstmals geöffnet wird. Die Struktur kann bei späteren Aufrufen der Funktion *OpenFile* dazu verwendet werden, sich auf die geöffnete Datei zu beziehen. Das Feld *szPathName* dieser Datenstruktur enthält Zeichen aus dem OEM-Zeichensatz.

Style: Bezeichnet die auszuführende Operation. Die folgenden Werte können mit dem bitweisen **or**-Operator kombiniert werden.

Wert	Bedeutung
of_Cancel	Fügt einen Aktionsschalter für *Abbruch* in das Dialogfenster *of_Prompt* ein. Die Betätigung des Abbruchschalters weist *OpenFile* an, eine Fehlermeldung `Datei nicht gefunden` zurückzugeben.
of_Create	Weist *OpenFile* an, eine neue Datei zu erstellen. Falls die Datei schon vorhanden ist, wird sie auf die Länge 0 gekürzt.
of_Delete	Löscht die Datei.
of_Exist	Öffnet die Datei und schließt sie wieder. Wird für eine Prüfung auf die Existenz einer Datei verwendet.
of_Parse	Füllt die Datenstruktur *TOFStruct*, führt aber keine andere Aktion aus.
of_Prompt	Zeigt ein Dialogfenster an, wenn die angesprochene Datei nicht existiert. Das Dialogfenster informiert den Benutzer, daß Windows die Datei nicht finden kann und fordert den Benutzer auf, eine Diskette mit der Datei in Laufwerk A einzulegen.
of_Read	Öffnet die Datei nur zum Lesen.
of_ReadWrite	Öffnet die Datei zum Lesen und Schreiben.
of_ReOpen	Öffnet die Datei unter Verwendung der Informationen im Datenpuffer für erneutes Öffnen.

Wert	Bedeutung
of_Share_Compact	Öffnet eine Datei im Kompatibilitätsmodus und ermöglicht dadurch das Öffnen der Datei, sooft es nötig ist. *OpenFile* kann nicht ausgeführt werden, wenn die Datei von einem anderen Prozeß im Kompatibilitätsmodus geöffnet wurde.
of_Share_Deny_None	Öffnet die Datei, ohne anderen Prozessen den Lese- oder Schreibzugriff auf diese Datei zu verweigern. *OpenFile* kann nicht ausgeführt werden, wenn die Datei von einem anderen Prozeß im Kompatibilitätsmodus geöffnet wurde.
of_Share_Deny_Read	Öffnet die Datei und verweigert anderen Prozessen den Lesezugriff auf die Datei. *OpenFile* kann nicht ausgeführt werden, wenn die Datei im Kompatibilitätsmodus oder für Lesezugriff von einem anderen Prozeß geöffnet wurde.
of_Share_Deny_Write	Öffnet die Datei und verweigert anderen Prozessen Schreibzugriff auf die Datei. *OpenFile* kann nicht ausgeführt werden, wenn die Datei im Kompatibilitätsmodus oder für Schreibzugriff von einem anderen Prozeß geöffnet wurde.
of_Share_Exclusive	Öffnet die Datei im Exklusivmodus und verweigert anderen Prozessen sowohl Lese- als auch Schreibzugriff auf die Datei. *OpenFile* kann nicht ausgeführt werden, wenn die Datei in irgendeinem anderen Modus für Lese- oder Schreibzugriff geöffnet wurde, selbst vom aktuellen Prozeß.
of_Verify	Überprüft, ob Datum und Zeit der Datei dieselben sind, wie beim vorherigen Öffnen. Dieser Wert ist zur zusätzlichen Überprüfung von schreibgeschützten Dateien nützlich.
of_Write	Öffnet die Datei nur zum Schreiben.

Rückgabewert Bei erfolgreicher Ausführung wird ein DOS-Datei-Handle zurückgeliefert, ansonsten -1.

Hinweis Wenn der Parameter *FileName* nur einen Dateinamen und die Namenserweiterung angibt, sucht die Funktion eine entsprechende Datei in den folgenden Verzeichnissen:

1. Im aktuellen Verzeichnis.

2. Im Windows-Verzeichnis (dem Verzeichnis, das WIN.COM enthält); die Funktion *GetWindowsDirectory* ermittelt den Pfadnamen dieses Verzeichnisses.

3. Im Windows-Systemverzeichnis (dem Verzeichnis, das Systemdateien wie KERNEL.EXE enthält); die Prozedur *GetSystemDirectory* ermittelt den Pfadnamen dieses Verzeichnisses.

4. In den Verzeichnissen, die in der Environment-Variablen PATH aufgelistet sind.

5. In den Verzeichnissen, die in einem Netz im Befehl MAP angegeben wurden. Windows durchsucht die Verzeichnisse in der angegebenen Reihenfolge. *FileName* darf keine Platzhalterzeichen enthalten. Um die Datei nach dem Gebrauch zu schließen, sollte die Anwendung die Funktion *_lclose* aufrufen.

OpenIcon

Syntax
```
function OpenIcon(Wnd: HWnd): Bool;
```

Beschreibung Diese Funktion aktiviert ein zum Symbol verkleinertes Fenster und zeigt es an. Windows stellt es in ursprünglicher Größe und an der ursprünglichen Position wieder her.

Wnd: Bezeichnet das Fenster.

Rückgabewert Bei erfolgreicher Ausführung der Funktion wird *True* zurückgeliefert, ansonsten *False*.

OpenSound

Syntax
```
function OpenSound: Integer;
```

Beschreibung Diese Funktion greift auf das Klangausgabegerät zu und verhindert, daß es nachfolgend von anderen Anwendungen angesteuert wird. Diese Funktion hat keine Parameter.

Rückgabewert Bei erfolgreicher Ausführung wird die Anzahl der verfügbaren Töne zurückgeliefert. Wenn das Klangausgabegerät bereits verwendet wird, ist der Rückgabewert *s_SerDVNA*, wenn nicht genügend Speicherplatz verfügbar ist, ist er *S_SerOFM*.

OutputDebugString

Syntax
```
procedure OutputDebugString(OutputString: PChar);
```

Beschreibung Diese Prozedur übergibt eine Debuggerbotschaft an den Debugger, wenn dieser vorhanden ist, andernfalls an die serielle Standardschnittstelle (AUX).

OutputString: Zeigt auf einen nullterminierten String.

Hinweis Diese Prozedur sichert alle Register. Sie ist nur in der Debugger-Version von Windows verfügbar.

PaintRgn

Syntax
```
function PaintRgn(DC: HDC; Rgn: HRgn): Bool;
```

Beschreibung Diese Funktion füllt die vom Parameter *Rgn* bestimmte Region mit dem ausgewählten Pinsel aus.

DC: Bezeichnet den Gerätekontext, der die Region enthält.

Rgn: Bezeichnet die Region, die ausgefüllt werden soll. Die Koordinaten für die angegebene Region sind in gerätespezifischen Einheiten angegeben.

Rückgabewert Bei erfolgreicher Ausführung wird *True* zurückgeliefert, ansonsten *False*.

PaletteIndex

Syntax
```
function PaletteIndex(PaletteIndex: Integer): TColorRef;
```

Beschreibung Diese Inline-Funktion akzeptiert einen Index zu einem virtuellen Paletteneintrag und gibt einen Wert zurück, der aus einer 1 im höherwertigen Byte und dem Paletteneintragsindex im niederwertigen Byte besteht. Dieser wird Paletteneintragsbezeichner genannt. Eine Anwendung, die eine Farbpalette benutzt, kann diesen Bezeichner für Funktionen, die eine Farbe erwarten, anstelle eines expliziten RGB-Wertes benutzen. Dies erlaubt der Funktion, die Farbe im angegebenen Paletteneintrag zu verwenden.

PaletteIndex: Gibt einen Index zu dem Paletteneintrag an, der die Farbe enthält, die für eine Grafikoperation benutzt werden soll.

Rückgabewert Der Rückgabewert ist ein Indexbezeichner zu einer virtuellen Palette. Bei Verwendung einer virtuellen Palette kann eine Anwendung anstelle eines expliziten RGB-Wertes diesen Bezeichner für GDI-Funktionen einsetzen, die eine Farbe anfordern.

PaletteRGB

Syntax
```
function PaletteRGB(R, G, B: Byte): LongInt;
```

Beschreibung Diese Inline-Funktion nimmt drei Werte entgegen, die die relativen Intensitäten von Rot, Grün und Blau darstellen, und

gibt einen Wert zurück, der im höherwertigen Byte aus einer 2 und in den drei niederwertigen Bytes aus einem RGB-Wert besteht – genannt palettenbezogener RGB-Bezeichner. Bei der Benutzung einer Farbpalette kann eine Anwendung diesen Bezeichner anstelle eines expliziten RGB-Werts an Funktionen übergeben, die eine Farbe anfordern.

Für Ausgabegeräte, die virtuelle Paletten unterstützen, gleicht Windows einen palettenbezogenen RGB-Wert an den nächsten Farbwert in der virtuellen Palette des Gerätekontexts an, als ob die Anwendung einen Index zu diesem Palleteneintrag angegeben hätte.

Wenn ein Ausgabegerät keine Systempalette unterstützt, benutzt Windows den palettenbezogenen RGB-Wert, als wäre er ein von der Funktion *RGB* zurückgegebener konventioneller *RGB*-LongInt.

Red: Bestimmt die Intensität des roten Farbwerts.

Green: Bestimmt die Intensität des grünen Farbwerts.

Blue: Bestimmt die Intensität des blauen Farbwerts.

Rückgabewert Es wird ein palettenbezogener RGB-Wert zurückgeliefert.

PatBlt

Syntax
```
function PatBlt(DC: HDC; X, Y, Width, Height: Integer;
Rop: LongInt): Bool;
```

Beschreibung Diese Funktion erzeugt auf dem angegebenen Gerät ein Bitmuster.

Das Muster ist eine Kombination des ausgewählten Pinsels mit dem Muster, das sich schon auf dem Gerät befindet. Der vom Parameter *Rop* angegebene Rasteroperationscode legt fest, wie die Muster kombiniert werden sollen.

DC: Bezeichnet den Gerätekontext.

X: Bestimmt die virtuelle x-Koordinate der oberen linken Ecke des Rechtecks, das das Muster aufnehmen soll.

Y: Bestimmt die virtuelle y-Koordinate der oberen linken Ecke des Rechtecks, das das Muster aufnehmen soll.

Width: Bestimmt die Breite des Rechtecks (in virtuellen Einheiten), das das Muster aufnehmen soll.

Height: Bestimmt die Höhe des Rechtecks (in virtuellen Einheiten), das das Muster aufnehmen soll.

Rop: Der Parameter *Rop* bestimmt den Rasteroperationscode. Möglich sind die in folgender Tabelle aufgeführten Werte:

Code	Beschreibung
PatCopy	Kopiert das vorgegebene Muster in das Bitmap.
PatInvert	Kombiniert Ziel-Bitmap und Muster unter Verwendung des logischen **xor**-Operators.
DSTInvert	Erzeugt ein invertiertes Bitmap.
Blackness	Erzeugt eine völlig geschwärzte Ausgabe.
Whiteness	Erzeugt eine völlig weiße Ausgabe.

Rückgabewert Der Rückgabewert ist *True*, wenn das Bitmuster gezeichnet wurde, andernfalls ist er *False*.

Hinweis Die Werte von *Rop* für diese Funktion sind eine begrenzte Untermenge der insgesamt 256 ternären Rasteroperationscodes; insbesondere kann ein Operationscode, der sich auf eine Quelle bezieht, nicht verwendet werden.

Nicht alle Geräte unterstützen die Funktion *PatBlt*. Weitere Informationen finden Sie bei der Beschreibung der Funktion *GetDeviceCaps* sowie in der Tabelle.

PeekMessage

Syntax
```
function PeekMessage(var Msg: TMsg; Wnd: HWnd;
MsgFilterMin, MsgFilterMax, RemoveMsg: Word): Bool;
```

Beschreibung Diese Funktion durchsucht die Anwendungs-Warteschlange nach einer Botschaft und legt die Botschaft (falls vorhanden) in *Msg* ab. Im Gegensatz zur Funktion *GetMessage* wartet *PeekMessage* nicht darauf, daß die Botschaft in die Warteschlange gestellt wird, bevor sie zurückkehrt. Sie übergibt jedoch die Steuerung (wenn das Flag *NoYield* nicht gesetzt ist) und gibt die Steuerung nach dieser Übergabe solange nicht zurück, bis Windows sie an die Anwendung zurückgibt. *PeekMessage* ruft innerhalb des durch die Parameter *MsgFilterMin* und *MsgFilterMax* bestimmten Bereichs von Botschaftswerten nur Botschaften ab, die dem durch den Parameter *Wnd* bezeichneten Fenster oder dessen von der Funktion *IsChild* bezeichneten untergeordneten Fenstern zugeordnet sind.

Wenn *Wnd* 0 ist, ruft *PeekMessage* Botschaften für jedes Fenster ab, das zur aufrufenden Anwendung gehört, die den Aufruf vornimmt. *PeekMessage* ruft keine Botschaften für Fenster ab, die zu anderen Anwendungen gehören. Wenn *Wnd* -1 ist, gibt *PeekMessage* nur Botschaften mit einem *Wnd* von 0 zurück, die von der Funktion *PostAppMessage* übermittelt werden. Wenn sowohl *MsgFilterMin* als auch *MsgFilterMax* 0 sind, gibt *PeekMessage* alle verfügbaren Botschaften zurück (es wird keine Bereichsfilterung durchgeführt).

Die Flags *wm_KeyFirst* und *wm_KeyLast* können als Filter-

werte zum Abrufen aller Tastaturbotschaften benutzt werden; die Flags *wm_MouseFirst* und *wm_MouseLast* können zum Abrufen aller Mausbotschaften benutzt werden.

Msg: TMsg-Datenstruktur, die die Botschaftsinformation aus der Windows-Anwendungs-Warteschlange enthält.

Wnd: Bezeichnet das Fenster, dessen Botschaften überprüft werden sollen.

MsgFilterMin: Bestimmt den Wert der untersten Botschafts-ID, die überprüft werden soll.

MsgFilterMax: Bestimmt den Wert der höchsten Botschafts-ID, die überprüft werden soll.

RemoveMsg: Bestimmt eine Kombination der Flags, die nachfolgend beschrieben sind. *pm_NoYield* kann entweder mit *pm_NoRemove* oder mit *pm_Remove* kombiniert werden.

Wert	Bedeutung
pm_NoRemove	Botschaften werden nach der Bearbeitung durch *PeekMessage* nicht aus der Botschafts-Warteschlange gelöscht.
pm_NoYield	Hindert die aktuelle Task daran, zu unterbrechen und Systemressourcen an eine andere Task abzugeben.
pm_Remove	Botschaften werden nach der Bearbeitung durch *PeekMessage* aus der Warteschlange gelöscht.

Rückgabewert Der Rückgabewert ist *True*, wenn eine Botschaft verfügbar ist, ansonsten ist er *False*.

Hinweis *PeekMessage* löscht keine Botschaften *wm_Paint* aus der Warteschlange, diese bleiben dort bis zur Bearbeitung. Ihr Programm sollte Aufrufe von *GetMessage*, *PeekMessage* und *WaitMessage* enthalten, da dies die einzige Möglichkeit ist, die Steuerung an andere Anwendungen zu übergeben.

Wenn *GetMessage*, *PeekMessage* und *WaitMessage* die Steuerung an andere Anwendungen abgeben, können sich die Stack- und Datensegmente der Anwendung, die die Funktion aufruft, im Arbeitsspeicher verschieben, um sich an den veränderten Speicherbedarf anderer Anwendungen anzupassen.

Wenn die Anwendung Zeiger auf Objekte im Daten- oder Stack-Segment gespeichert hat (globale oder lokale Variable) und deren Sperren aufgehoben sind, können diese durch einen Aufruf von *GetMessage*, *PeekMessage* oder *WaitMessage* ungültig werden. Der Parameter *Msg* der aufgerufenen Funktion bleibt in jedem Fall gültig.

Querverweis *GetMessage, WaitMessage*

Pie

Syntax

```
function Pie(DC: HDC; X1, Y1, X2, Y2, X3, Y3, X4, Y4:
Integer): Bool;
```

Beschreibung

Diese Funktion zeichnet ein Tortendiagramm, indem sie einen Ellipsenbogen zeichnet, dessen Mittelpunkt und zwei Endpunkte durch Linien miteinander verbunden werden. Der Mittelpunkt des Bogens ist der Mittelpunkt des umgrenzenden Rechtecks, das die Parameter *X1*, *Y1*, *X2* und *Y2* vorgeben. Die Start- und Endpunkte des Bogens sind durch die Parameter *X3*, *Y3*, *X4* und *Y4* bezeichnet. Der Bogen wird mit dem ausgewählten Stift entgegen dem Uhrzeigersinn gezeichnet und zusätzlich werden die beiden Endpunkte des Bogens mit seinem Mittelpunkt durch zwei Linien verbunden. Das Tortenstück wird mit dem ausgewählten Pinsel ausgefüllt.

Wenn *X3* gleich *X4* und *Y3* gleich *Y4* ist, dann ist das Ergebnis eine Ellipse mit einer einzigen Linie vom Mittelpunkt der Ellipse zum Punkt (*X3*,*Y3*).

DC: Bezeichnet den Gerätekontext.

X1: Bestimmt die virtuelle x-Koordinate der oberen linken Ecke des umgebenden Rechtecks.

Y1: Bestimmt die virtuelle y-Koordinate der oberen linken Ecke des umgebenden Rechtecks.

X2: Bestimmt die virtuelle x-Koordinate der unteren rechten Ecke des umgebenden Rechtecks.

Y2: Bestimmt die virtuelle y-Koordinate der unteren rechten Ecke des umgebenden Rechtecks.

X3: Bestimmt die virtuelle x-Koordinate des Startpunkts des Bogens. Dieser Punkt muß nicht exakt auf dem Bogen liegen.

Y3: Bestimmt die virtuelle y-Kordinate des Startpunkts des Bogens. Dieser Punkt muß nicht exakt auf dem Bogen liegen.

X4: Bestimmt die virtuelle x-Koordinate des Endpunkts des Bogens. Dieser Punkt muß nicht exakt auf dem Bogen liegen.

Y4: Bestimmt die virtuelle y-Koordinate des Endpunkts des Bogens. Dieser Punkt muß nicht exakt auf dem Bogen liegen.

Rückgabewert

Der Rückgabewert ist *True*, wenn das Tortendiagramm gezeichnet wurde, andernfalls ist er *False*.

Hinweis

Die Breite des Rechtecks, die durch die Differenz *X2 - X1* festgelegt ist, darf 32767 Einheiten nicht überschreiten. Diese Begrenzung gilt auch für die Höhe des Rechtecks (*Y2 - Y1*).

Die aktuelle Position wird von dieser Funktion weder verwendet noch aktualisiert.

PlayMetaFile

Syntax
```
function PlayMetaFile(DC: HDC; MF: THandle): Bool;
```

Beschreibung Diese Funktion spielt den Inhalt des bezeichneten Metafiles auf dem angegebenen Gerät ab. Das Metafile kann beliebig oft wiedergegeben werden.

DC: Bezeichnet den Gerätekontext des Ausgabegerätes.

MF: Bezeichnet das Metafile.

Rückgabewert Der Rückgabewert ist *True*, wenn die Funktion erfolgreich ist, andernfalls ist er *False*.

PlayMetaFileRecord

Syntax
```
procedure PlayMetaFileRecord(DC: HDC; var Handletable:
THandleTable; var MetaRecord: TMetaRecord; Handles:
Word);
```

Beschreibung Diese Prozedur spielt ein Metafile-Record durch Ausführung des GDI-Funktionsaufrufes ab, der in dem Metafile-Record enthalten ist.

DC: Bezeichnet den Gerätekontext des Ausgabegeräts.

Handletable: Eine Objekt-Handle-Tabelle (*THandleTable*-Struktur), die für die Metafile-Wiedergabe benutzt werden soll.

MetaRecord: Wiederzugebender *TMetaRecord* des Metafile.

Handles: Bestimmt die Anzahl der Handles in der Handle-Tabelle.

Hinweis Eine Anwendung benutzt diese Prozedur üblicherweise in Verbindung mit der Funktion *EnumMetafile*, um ein Metafile erst zu modifizieren und dann wiederzugeben.

Querverweis *EnumMetaFile*

Polygon

Syntax
```
function Polygon(DC: HDC; var Points; Count: Integer):
Bool;
```

Beschreibung Diese Funktion zeichnet ein Polygon, das aus zwei oder mehr Punkten (Eckpunkten) besteht, die durch Linien verbunden sind. Diese Polygone werden durch Verwendung des aktuellen Polygonfüllmodus ausgefüllt.

Eine Beschreibung der Polygonfüllmodi finden Sie in Kapitel 1 »Windows-Stile und -Konstanten« im Abschnitt »Polygonfüllmodi«. Das Polygon wird, falls dies erforderlich ist,

durch Zeichnen einer Linie vom letzten Eckpunkt zum ersten automatisch geschlossen.

DC: Bezeichnet den Gerätekontext.

Points: Ein Array von Punkten, die die Eckpunkte des Polygons bezeichnen. Jeder Punkt im Array ist eine *TPoint*-Datenstruktur.

Count: Bestimmt die Zahl von Eckpunkten, die im Array beschrieben sind.

Rückgabewert Der Rückgabewert ist *True*, wenn die Funktion erfolgreich war, andernfalls ist er *False*.

Hinweis Die aktuelle Position wird von dieser Funktion weder benutzt noch aktualisiert.

Der aktuelle Polygonfüllmodus kann mit der Funktion *GetPolyFillMode* abgerufen und mit *SetPolyFillMode* gesetzt werden.

Querverweis *SetPolyFillMode*

Polyline

Syntax
```
function Polyline(DC: HDC; var Points; Count: Integer): Bool;
```

Beschreibung Diese Funktion zeichnet eine Folge von Linienzügen, die die vom Parameter *Points* bezeichneten Punkte der Reihe nach verbinden. Das Ergebnis ist dasselbe, als wären die Funktionen *MoveTo* und *LineTo* benutzt worden. Die aktuelle Position wird von der Funktion *Polyline* weder benutzt noch aktualisiert.

DC: Bezeichnet den Gerätekontext.

Points: Ein Array von Punkten, die verbunden werden sollen. Jeder Punkt in dem Array ist eine *TPoint*-Datenstruktur.

Count: Bestimmt die Zahl der Punkte im Array. Der Parameter *Count* muß mindestens 2 sein.

Rückgabewert Der Rückgabewert ist *True*, wenn die Liniensegmente gezeichnet wurden, andernfalls ist er *False*.

Hinweis *Polyline* zeichnet die Linien mit dem ausgewählten Stift.

PolyPolygon

Syntax
```
function PolyPolygon(DC: HDC; var Points; var
PolyCounts; Count: Integer): Bool;
```

Beschreibung Diese Funktion erzeugt eine Reihe von geschlossenen Polygonen. Die Polygone werden im aktuellen Polygonfüllmodus ausgefüllt. Eine Beschreibung der Polygonfüllmodi finden Sie

in Kapitel 1 »Windows-Stile und -Konstanten« im Abschnitt »Polygonfüllmodi«. Die Polygone können sich überlappen, dies ist aber nicht erforderlich.

DC: Bezeichnet den Gerätekontext.

Points: Ein Array von *TPoint*-Datenstrukturen, die der Reihe nach die Eckpunkte mehrerer Polygone definieren. Jedes Polygon muß geschlossen sein. Im Gegensatz zu Polygonen, die mit der Funktion *Polygon* erzeugt werden, werden mit *PolyPolygon* erzeugte Polygone nicht automatisch geschlossen.

PolyCounts: Ein Array von Integerwerten, die für jedes Polygon die jeweilige Anzahl der Punkte im Array *Points* bezeichnet.

Count: Bestimmt die Gesamtzahl von Integerwerten im Array *PolyCounts*.

Rückgabewert Der Rückgabewert ist ungleich 0, wenn die Polygone gezeichnet wurden, andernfalls ist er 0.

PostAppMessage

Syntax
```
function PostAppMessage(Task: THandle; Msg, wParam:
Word; lParam: LongInt): Bool;
```

Beschreibung Diese Funktion übermittelt eine Botschaft an eine Anwendung, die von einem Task-Handle bezeichnet wird und kehrt dann zurück, ohne auf Bestätigung zu warten. Die Anwendung, die die Botschaft empfängt, ruft diese ab, indem sie eine der Funktionen *GetMessage* oder *PeekMessage* aufruft. Der Parameter *Wnd* der zurückgegebenen *TMsg*-Datenstruktur ist 0.

Task: Bezeichnet die Task, die die Botschaft empfangen soll. Die Funktion *GetCurrentTask* gibt dieses Handle zurück.

Msg: Bestimmt den Typ der übermittelten Botschaft.

wParam: Bestimmt zusätzliche Botschaftsinformationen.

lParam: Bestimmt zusätzliche Botschaftsinformationen.

Rückgabewert Der Rückgabewert ist *True*, wenn die Botschaft abgesetzt wurde, andernfalls ist er *False*.

Querverweis *GetCurrentTask, GetMessage, PeekMessage*

PostMessage

Syntax
```
function PostMessage(Wnd: HWnd; Msg, wParam: Word;
lParam: LongInt): Bool;
```

Beschreibung Diese Funktion plaziert eine Botschaft in die Anwendungs-Warteschlange eines Fensters und kehrt dann zurück, ohne

darauf zu warten, daß das zugehörige Fenster die Botschaft bearbeitet. Die übermittelte Botschaft kann durch einen Aufruf einer der Funktionen *GetMessage* oder *PeekMessage* abgerufen werden.

Wnd: Bezeichnet das Fenster, das die Botschaft empfangen soll. Wenn der Parameter *Wnd* $FFFF ist, wird die Botschaft an alle überlappten oder Pop-Up-Fenster im System gesendet. Die Botschaft wird nicht an untergeordnete Fenster gesendet.

Msg: Bestimmt den Typ der übermittelten Botschaft.

wParam: Bestimmt zusätzliche Botschaftsinformationen.

lParam: Bestimmt zusätzliche Botschaftsinformationen.

Rückgabewert Der Rückgabewert ist *True*, wenn die Botschaft übermittelt wurde, andernfalls ist er *False*.

Hinweis Eine Anwendung sollte die Funktion *PostMessage* nie verwenden, um eine Botschaft an ein Steuerelement zu senden. Wenn ein System, in dem Windows läuft, für Expanded Memory (EMS) konfiguriert ist und eine Anwendung mit der Funktion *PostMessage* eine Botschaft mit verknüpften Daten (auf die der Parameter *lParam* zeigt) an eine zweite Anwendung sendet, muß die erste Anwendung diese Daten in dem globalen Speicherbereich plazieren, der von der Funktion *GlobalAlloc* mit dem Flag *gmem_Lower* reserviert wurde. Diese Speicherreservierung ist nur notwendig, wenn *lParam* einen Zeiger enthält.

Im Gegensatz zu anderen Windows-Funktionen kann eine Anwendung *PostMessage* auf Hardware-Interrupt-Ebene aufrufen.

PostQuitMessage

Syntax
```
procedure PostQuitMessage(ExitCode: Integer);
```

Beschreibung Diese Prozedur informiert Windows darüber, daß die Anwendung ihre Ausführung beenden möchte. Sie wird üblicherweise als Antwort auf eine Botschaft *wm_Destroy* benutzt.

PostQuitMessage übermittelt der Anwendung eine Botschaft *wm_Quit*, die dem System mitteilt, daß die Anwendung demnächst beenden möchte, und kehrt sofort zurück.

Wenn die Anwendung die Botschaft *wm_Quit* empfängt, sollte sie die Botschaftsschleife in der Hauptfunktion verlassen und die Steuerung an Windows zurückgeben. Der Exit-Code, der an Windows zurückgegeben wird, muß der Parameter *wParam* der Botschaft *wm_Quit* sein.

ExitCode: Bestimmt einen Exit-Code der Anwendung. Er wird ebenso wie der Parameter *wParam* in der Botschaft *wm_Quit* benutzt.

PtInRect

Syntax `function PtInRect(var Rect: TRect; Point: TPoint): Bool;`

Beschreibung Diese Funktion bestimmt, ob der angegebene Punkt innerhalb eines vorgegebenen Rechtecks liegt. Ein Punkt liegt innerhalb eines Rechtecks, wenn er auf dem linken oder oberen Rand oder innerhalb aller vier Kanten liegt. Ein Punkt auf dem rechten oder unteren Rand liegt außerhalb des Rechtecks.

Rect: Eine *TRect*-Datenstruktur, die das Rechteck enthält.

Point: Eine *TPoint*-Datenstruktur, die den Punkt enthält.

Rückgabewert Der Rückgabewert ist *True*, wenn der Punkt innerhalb des angegebenen Rechtecks liegt, andernfalls ist er *False*.

PtInRegion

Syntax `function PtInRegion(Rgn: HRgn; X, Y: Integer): Bool;`

Beschreibung Diese Funktion ermittelt, ob der durch die Parameter *X* und *Y* angegebene Punkt in der angegebenen Region liegt.

Rgn: Bezeichnet die Region, die überprüft werden soll.

X: Bestimmt die virtuelle x-Koordinate des Punktes.

Y: Bestimmt die virtuelle y-Koordinate des Punktes.

Rückgabewert Der Rückgabewert ist *True*, wenn der Punkt in der Region liegt, andernfalls ist er *False*.

PtVisible

Syntax `function PtVisible(DC: HDC; X, Y: Integer): Bool;`

Beschreibung Diese Funktion ermittelt, ob der angebene Punkt innerhalb der Clipping-Region des vorgegebenen Gerätekontexts liegt.

DC: Bezeichnet den Gerätekontext.

X: Bestimmt die virtuelle x-Koordinate des Punktes.

Y: Bestimmt die virtuelle y-Koordinate des Punktes.

Rückgabewert Der Rückgabewert ist *True*, wenn der Punkt innerhalb der Clipping-Region liegt, andernfalls ist er *False*.

ReadComm

Syntax `function ReadComm(Cid: Integer; Buf: PChar; Size: Integer): Integer;`

Beschreibung Diese Funktion liest die durch den Parameter *Size* bestimmte Anzahl der Zeichen aus der durch den Parameter *Cid* bezeichneten Schnittstelle und kopiert die Zeichen in den Puffer, auf den der Parameter *Buf* zeigt.

Cid: Bestimmt die Schnittstelle, von der gelesen werden soll. Die Funktion *OpenComm* gibt diesen Wert zurück.

Buf: Zeigt auf den Puffer, der die gelesenen Zeichen empfangen soll.

Size: Bestimmt die Zahl der zu lesenden Zeichen.

Rückgabewert Der Rückgabewert gibt die Zahl der tatsächlich gelesenen Zeichen an. Er ist kleiner als die durch *Size* angegebene Zahl, wenn die Anzahl der Zeichen in der Empfangs-Warteschlange kleiner als diese ist. Ist er gleich *Size*, können zusätzliche Zeichen für das Gerät in die Warteschlange gestellt worden sein. Sind keine Zeichen vorhanden, ist der Rückgabewert 0.

Tritt ein Fehler auf, wird der Rückgabewert auf einen Wert kleiner 0 gesetzt, wobei der Absolutbetrag die Zahl der tatsächlich gelesenen Zeichen darstellt. Die Fehlerursache kann mit der Funktion *GetCommError* bestimmt werden, indem Fehlercode und -status abgerufen werden. Da Fehler auftreten können, wenn keine Bytes vorhanden sind, der Rückgabewert also 0 ist, sollte die Funktion *GetCommError* auch in diesem Fall benutzt werden, um sicherzustellen, daß kein Fehler aufgetreten ist.

Für parallele I/O-Anschlüsse ist der Rückgabewert immer 0.

Querverweis *GetCommError, OpenComm*

RealizePalette

Syntax
```
function RealizePalette(DC: HDC): Word;
```

Beschreibung Diese Funktion bildet die Systempaletteneinträge in der aktuell ausgewählten virtuellen Palette in einen Gerätekontext ab.

Eine virtuelle Farbpalette fungiert als Puffer zwischen farbintensiven Anwendungen und dem System. Sie erlaubt einer Anwendung, so viele Farben wie nötig zu benutzen, ohne daß dadurch die eigene Farbdarstellung oder Farben, die von anderen Fenstern dargestellt werden, überlagert werden.

Wenn ein Fenster den Fokus hat und *RealizePalette* aufruft, stellt Windows sicher, daß alle angeforderten Farben bis zur Maximalzahl gleichzeitig verfügbarer Farben auf dem Bildschirm dargestellt werden. Zusätzliche Farben zeigt es an, indem diese an verfügbare Farben angeglichen werden. Außerdem gleicht Windows die Farben inaktiver Fenster, die *RealizePalette* aufrufen, so weit wie möglich an die verfügbaren Farben an, womit nicht erwünschte Veränderungen der Farben, die in

inaktiven Fenstern dargestellt werden, deutlich verringert werden.

DC: Bezeichnet den Gerätekontext.

Rückgabewert Der Rückgabewert bestimmt, wieviele Einträge in der virtuellen Palette auf Einträge in der Systempalette abgebildet wurden. Das entspricht der Zahl der Einträge, die diese Funktion neu abgebildet hat, um sich den Änderungen in der Systempalette seit der letzten Realisierung der virtuellen Palette anzupassen.

Rectangle

Syntax
```
function Rectangle(DC: HDC; X1, Y1, X2, Y2: Integer): Bool;
```

Beschreibung Diese Funktion zeichnet ein Rechteck. Das Innere des Rechtecks wird mit dem gewählten Pinsel ausgefüllt und die Umrandung mit dem gewählten Stift gezeichnet.

DC: Bezeichnet den Gerätekontext.

X1: Bestimmt die virtuelle x-Koordinate der oberen linken Ecke des Rechtecks.

Y1: Bestimmt die virtuelle y-Koordinate der oberen linken Ecke des Rechtecks.

X2: Bestimmt die virtuelle x-Koordinate der unteren rechten Ecke des Rechtecks.

Y2: Bestimmt die virtuelle y-Koordinate der unteren rechten Ecke des Rechtecks.

Rückgabewert Der Rückgabewert ist *True*, wenn das Rechteck gezeichnet wurde, andernfalls ist er *False*.

Hinweis Die Breite des Rechtecks *X2 - X1* darf 32767 Einheiten nicht überschreiten. Dieser Grenzwert gilt auch für die Höhe des Rechtecks (*Y2 - Y1*).

Die aktuelle Position wird von dieser Funktion weder verwendet noch aktualisiert.

RectInRegion

Syntax
```
function RectInRegion(Rgn: HRgn; var Rect: TRect): Bool;
```

Beschreibung Diese Funktion bestimmt, ob ein Teil des durch den Parameter *Rect* vorgegebenen Rechtecks innerhalb der Grenzen einer Region liegt, die durch den Parameter *Rgn* bezeichnet ist.

Region: Bezeichnet die Region.

Rect: *TRect*-Datenstruktur die die Rechteckkoordinaten enthält.

Rückgabewert Der Rückgabewert ist *True*, wenn ein Teil des angegebenen Rechtecks innerhalb der Grenzen der Region liegt, andernfalls ist er *False*.

RectVisible

Syntax
```
function RectVisible(DC: HDC; var Rect: TRect): Bool;
```

Beschreibung Diese Funktion stellt fest, ob ein Teil des gegebenen Rechtecks innerhalb der Clipping-Region des angegebenen Bildschirmkontexts liegt.

DC: Bezeichnet den Gerätekontext.

Rect: Eine *TRect*-Datenstruktur, die die virtuellen Koordinaten des Rechtecks enthält.

Rückgabewert Der Rückgabewert ist *True*, wenn ein Stück des angegebenen Rechtecks innerhalb der Clipping-Region liegt, andernfalls ist er *False*.

RegisterClass

Syntax
```
function RegisterClass(var WndClass : TWndClass): Bool;
```

Beschreibung Diese Funktion registriert eine Fensterklasse zur nachfolgenden Verwendung in Aufrufen der Funktion *CreateWindow*. Die Fensterklasse hat die Attribute, die durch die Datenstruktur definiert werden, auf die der Parameter *WndClass* zeigt.

Sollen zwei Klassen mit demselben Namen registriert werden, mißlingt der zweite Versuch und die Information für diese Klasse wird ignoriert.

WndClass: Eine *TWndClass*-Datenstruktur. Die Struktur muß mit den geeigneten Klassenattributen gefüllt sein, bevor sie an die Funktion übergeben wird. Im folgenden Abschnitt sind Details aufgeführt.

Rückgabewert Der Rückgabewert ist *True*, wenn die Klasse registriert wurde, andernfalls ist er *False*.

Callback-Funktion
```
function WndProc (Wnd: HWnd; wMsg, wParam: Word;
lParam: LongInt): Bool; export;
```

WndProc: Ein Platzhalter für den von der Anwendung anzugebenden Funktionsnamen.

hWnd: Bezeichnet das Fenster, das die Botschaft empfängt.

wMsg: Bestimmt die Botschaftsnummer.

wParam: Bestimmt zusätzliche, botschaftsabhängige Information.

lParam: Bestimmt zusätzliche, botschaftsabhängige Information.

Rückgabewert Die Fensterfunktion gibt das Ergebnis der Botschaftsverarbeitung zurück. Die möglichen Rückgabewerte hängen von der aktuell gesendeten Botschaft ab.

RegisterClipboardFormat

Syntax
```
function RegisterClipboardFormat(FormatName: PChar): Word;
```

Beschreibung Diese Funktion registriert ein neues Zwischenablageformat, auf dessen Bezeichnung der Parameter *FormatName* zeigt. Das registrierte Format kann in darauffolgenden Zwischenablagefunktionen als gültiges Format, in dem Daten übergeben werden können, benutzt werden. Es erscheint in der Formatliste der Zwischenablage.

FormatName: Zeigt auf einen nullterminierten String, der das neue Format benennt.

Rückgabewert Der Rückgabewert beschreibt das neu registrierte Format. Wenn der identische Formatname schon vorher registriert wurde (z.B. durch eine andere Anwendung), wird der Referenzzähler des Formats erhöht und derselbe Wert zurückgegeben, als wäre das Format erstmals registriert worden. Der Rückgabewert ist 0, wenn das Format nicht registriert werden kann.

Hinweis Der Formatwert, den die Funktion *RegisterClipboardFormat* zurückgibt, liegt im Bereich von $C000 bis $FFFF.

RegisterWindowMessage

Syntax
```
function RegisterWindowMessage(Str: PChar): Word;
```

Beschreibung Diese Funktion definiert eine neue Fensterbotschaft, die systemweit garantiert eindeutig ist. Der zurückgegebene Botschaftswert kann benutzt werden, wenn die Funktionen *SendMessage* oder *PostMessage* aufgerufen werden. *RegisterWindowMessage* wird üblicherweise für den Datenaustausch zwischen zwei zusammenarbeitenden Anwendungen benutzt.

Wird derselbe Botschaftsstring von zwei unterschiedlichen Anwendungen registriert, so wird derselbe Botschaftswert zurückgegeben. Die Botschaft bleibt registriert, bis der Benutzer die Windows-Sitzung beendet.

Str: Zeigt auf den Botschaftsstring, der registriert werden soll.

Rückgabewert Bei erfolgreicher Ausführung ist der Rückgabewert ein Word im Bereich zwischen $C000 bis $FFFF, andernfalls ist er 0.

Hinweis *RegisterWindowMessage* sollte nur benutzt werden, wenn dieselbe Nachricht von mehr als einer Anwendung verstanden werden muß. Um private Botschaften innerhalb einer Anwendung zu senden, kann eine Anwendung ein Word im Bereich von *wm_User* bis $BFFF verwenden.

ReleaseCapture

Syntax
```
procedure ReleaseCapture;
```

Beschreibung Hebt die exklusive Benutzung der Maus auf und stellt die normale Eingabeverarbeitung wieder her. Diese Funktion hat keine Parameter.

Hinweis Eine Anwendung ruft diese Prozedur nach einem Aufruf der Funktion *SetCapture* auf.

ReleaseDC

Syntax
```
function ReleaseDC(Wnd: HWnd; DC: HDC): Integer;
```

Beschreibung Diese Funktion hebt einen Gerätekontext auf und gibt ihn zur Benutzung durch andere Anwendungen frei. Die Wirkung von *ReleaseDC* hängt vom Typ des Gerätekontexts ab. Sie gibt nur allgemeine und Fenster-Gerätekontexte frei und hat keinen Einfluß auf Klassen- oder private Gerätekontexte.

Wnd: Bezeichnet das Fenster, dessen Gerätekontext aufgehoben werden soll.

DC: Bezeichnet den aufzuhebenden Gerätekontext.

Rückgabewert Der Rückgabewert ist 1, wenn der Gerätekontext aufgehoben wurde, andernfalls ist er 0.

Hinweis Die Anwendung muß *ReleaseDC* für jeden Aufruf von *GetWindowDC* und *GetDC*, der einen gemeinsamen Gerätekontext abruft, aufrufen.

Querverweis *GetDC, GetWindowDC*

RemoveFontResource

Syntax
```
function RemoveFontResource(Filename: PChar): Bool;
```

Beschreibung Diese Funktion löscht eine hinzugefügte Schriftressource aus der Datei, die durch den Parameter *Filename* benannt ist, oder aus der Windows-Schrifttabelle.

Filename: Zeigt auf einen String, der die Schriftressourcendatei benennt oder ein Handle zu einem geladenenen Modul enthält. Zeigt *Filename* auf den Dateinamen der Schriftressource, muß der String nullterminiert sein und dem DOS-Dateinamensformat entsprechen.

Wenn *Filename* ein Handle enthält, muß dieses im niederwertigen Word liegen und das höherwertige Word 0 sein.

Rückgabewert Der Rückgabewert ist *True*, wenn die Funktion erfolgreich war, andernfalls ist er *False*.

Hinweis Jede Anwendung, die Schriften aus der Windows-Schriftentabelle löscht oder ihr hinzufügt, sollte andere Fenster mit der Funktion *SendMessage* davon benachrichtigen. Dabei sollte der Parameter *Wnd* auf -1 gesetzt sein, um eine Botschaft *wm_FontChange* an alle Hauptfenster im System zu senden. *RemoveFontResource* löscht die Schriftressource nicht unmittelbar. Wenn es noch ausstehende Referenzen zu ihr gibt, bleibt sie geladen, bis mit der Funktion *DeleteObject* die letzte sich auf sie beziehende virtuelle Schrift gelöscht wurde.

Querverweis *DeleteObject, wm_FontChange*

RemoveMenu

Syntax
```
function RemoveMenu(Menu: HMenu; Position, Flags: Word): Bool;
```

Beschreibung Diese Funktion löscht einen Menüeintrag mit verbundenem Pop-up-Menü aus dem durch den Parameter *Menu* bezeichneten Menü. Dabei gibt sie das Handle auf das Pop-up-Menü nicht frei und erlaubt somit die Wiederverwendung des Menüs. Vor dem Aufruf von *RemoveMenu* sollte die Anwendung *GetSubMenu* aufrufen, um das Handle des Pop-up-Menüs abzurufen.

Menu: Bezeichnet das Menü, das verändert werden soll.

Position: Bestimmt die zu löschende Menüoption. Die Interpretation des Parameters *Position* hängt vom Wert des Parameters *Flags* ab.

Flags	Position
mf_ByCommand	Bestimmt die Befehls-ID der existierenden Menüoption.
mf_ByPosition	Bestimmt die Position der Menüoption. Die erste Option im Menü steht an Position 0.

Flags: Bestimmt, wie der Parameter *Position* interpretiert wird. Er muß entweder *mf_ByCommand* oder *mf_ByPosition* sein.

Rückgabewert Der Rückgabewert ist *True*, wenn die Funktion erfolgreich war, andernfalls ist er *False*.

Hinweis Nach jeder Änderung eines Menüs (unabhängig davon, ob es sich im angezeigten Fenster befindet oder nicht), sollte die Prozedur *DrawMenuBar* aufgerufen werden.

Querverweis *DrawMenuBar, GetSubMenu*

RemoveProp

Syntax
```
function RemoveProp(Wnd: HWnd; Str: PChar): THandle;
```

Beschreibung Diese Funktion löscht einen Eintrag aus der Eigenschaftsliste des angegebenen Fensters. Der String, der durch den Parameter *Str* bestimmt ist, bezeichnet den zu löschenden Eintrag.

RemoveProp gibt das Daten-Handle zurück, das mit dem String verbunden ist, so daß die Anwendung die Daten, die mit dem Handle verbunden sind, freigeben kann.

Wnd: Bezeichnet das Fenster, dessen Eigenschaftsliste geändert werden soll.

Str: Zeigt auf einen nullterminierten String oder auf ein Atom, das einen String bezeichnet. Wenn ein Atom angegeben ist, dann muß es vorher mittels der Funktion *AddAtom* erzeugt worden sein. Das Atom, ein 16-Bit-Wert, muß im niederwertigen Word von *Str* plaziert werden; das höherwertige Word muß 0 sein.

Rückgabewert Der Rückgabewert ist 0, wenn der String in der angegebenen Eigenschaftsliste nicht gefunden werden kann.

Hinweis Eine Anwendung muß die Daten-Handles, die gelöschten Einträgen aus einer Eigenschaftsliste zugeordnet sind, freigeben. Die Anwendung sollte nur diejenigen Eigenschaften löschen, die sie in die Eigenschaftsliste eingefügt hatte.

ReplyMessage

Syntax
```
procedure ReplyMessage(Reply: LongInt);
```

Beschreibung Diese Prozedur wird benutzt, um eine mit der Funktion *SendMessage* übergebene Botschaft zu beantworten, ohne die Steuerung an die von *SendMessage* aufgerufene Funktion zurückzugeben. Der Aufruf dieser Prozedur erlaubt es den Aufrufen von *SendMessage* und *ReplyMessage*, mit der Bearbeitung fortzufahren.

Normalerweise fährt eine Task, die *SendMessage* aufruft, um eine Botschaft an eine andere Task zu senden, solange nicht mit der Ausführung fort, bis die Fensterprozedur antwortet, die

Windows aufruft, um die Botschaft zu empfangen. Muß jedoch eine Task, die zum Empfangen einer Botschaft aufgerufen wurde, bestimmte Operationen ausführen, die die Steuerung abgeben könnten (etwa die Funktionen *MessageBox* oder *DialogBox*), könnte Windows blockiert werden. Der Grund liegt darin, daß die sendende Task Botschaften zur Ausführung und Verarbeitung braucht, diese aber nicht erhält, weil sie auf die Rückkehr von *SendMessage* wartet. Eine Anwendung kann dieses Problem umgehen, indem die Task, die die Botschaft empfängt, *ReplyMessage* aufruft, bevor sie eine Operation ausführt, die eine Abgabe der Steuerung bewirken könnte.

ReplyMessage hat keine Wirkung, wenn die Botschaft nicht durch *SendMessage* oder durch dieselbe Task gesendet wurde.

Reply: Das Ergebnis der Verarbeitung der Botschaft. Die möglichen Werte hängen von der tatsächlich gesendeten Botschaft ab.

ResizePalette

Syntax
```
function ResizePalette(Palette: HPalette; NumEntries: Word): Bool;
```

Beschreibung Diese Funktion ändert die Größe der virtuellen Palette, die durch den Parameter *Palette* bestimmt ist, in die Anzahl von Einträgen, die durch den Parameter *NumEntries* vorgegeben wird. Ruft eine Anwendung *ResizePalette* auf, um die Palette zu verkleinern, werden die Einträge, die in der Palette mit der neuen Größe zurückbleiben, nicht verändert. Ruft die Anwendung *ResizePalette* auf, um sie zu vergrößern, werden die zusätzlichen Paletteneinträge auf Schwarz gesetzt (die Rot-, Grün- und Blauwerte sind alle 0). Außerdem werden die Flags für alle zusätzlichen Einträge auf 0 gesetzt.

Palette: Bezeichnet die Palette, die verändert werden soll.

NumEntries: Bestimmt die Zahl der Einträge in der Palette, nachdem ihre Größe neu festgelegt wurde.

Rückgabewert Der Rückgabewert ist *True*, wenn die Größe der Palette erfolgreich geändert wurde, andernfalls ist er *False*.

RestoreDC

Syntax
```
function RestoreDC(DC: HDC; SavedDC: Integer): Bool;
```

Beschreibung Diese Funktion stellt den mit *DC* angegebenen Gerätekontext entsprechend dem mit *SavedDC* bezeichneten vorherigen Status wieder her. *RestoreDC* stellt durch Kopieren von Statusinformationen, die durch vorausgegangene Aufrufe der Funk-

tion *SaveDC* auf dem Kontext-Stack abgelegt wurden, den Gerätekontext wieder her.

Der Kontext-Stack kann Statusinformationen für mehrere Gerätekontexte enthalten. Wenn der durch *SavedDC* angegebene Kontext nicht oben auf dem Stack abgelegt ist, löscht *RestoreDC* jede Statusinformation zwischen dem durch *SavedDC* bezeichneten Gerätekontext und dem oberen Ende des Stacks, diese Informationen sind somit verloren.

DC: Bezeichnet den Gerätekontext.

SavedDC: Bestimmt den Gerätekontext, der wiederhergestellt werden soll.

Er kann ein Wert sein, der von einem vorangegangenen Aufruf der Funktion *SaveDC* zurückgegeben wurde. Wenn *SavedDC* -1 ist, wird der zuletzt gesicherte Gerätekontext wiederhergestellt.

Rückgabewert Der Rückgabewert ist *True*, wenn der angegebene Gerätekontext wiederhergestellt wurde, andernfalls ist er *False*.

RGB

Syntax
```
function RGB(R, G, B: Byte): LongInt;
```

Beschreibung Diese Inline-Funktion wählt aufgrund der gegebenen Parameter und der Farbwiedergabemöglichkeiten des Ausgabegeräts eine RGB-Farbe aus.

R: Bestimmt die Intensität des roten Farbfeldes.

G: Bestimmt die Intensität des grünen Farbfeldes.

B: Bestimmt die Intensität des blauen Farbfeldes.

Rückgabewert Der Rückgabewert ist die resultierende RGB-Farbe.

Hinweis Die Intensität für jedes Argument kann von 0 bis 255 reichen. Wenn alle drei Intensitätswerte mit 0 angegeben sind, ist das Ergebnis Schwarz. Wenn alle drei Intensitätswerte mit 255 angegeben werden, ist das Ergebnis Weiß.

Informationen über den Einsatz von Farbwerten in einer Farbpalette ist in den Beschreibungen der Inline-Funktionen *PaletteIndex* und *PaletteRGB* zu finden.

RoundRect

Syntax
```
function RoundRect(DC: HDC; X1, Y1, X2, Y2, X3, Y3:
Integer): Bool;
```

Beschreibung Diese Funktion zeichnet Rechtecke mit abgerundeten Ecken. Das Innere des Rechtecks wird mit dem gewählten Pinsel

ausgefüllt, die Umrandung wird mit dem ausgewählten Stift gezeichnet.

DC: Bezeichnet den Gerätekontext.

X1: Bestimmt die virtuelle x-Koordinate der oberen linken Ecke des Rechtecks.

Y1: Bestimmt die virtuelle y-Koordinate der oberen linken Ecke des Rechtecks.

X2: Bestimmt die virtuelle x-Koordinate der unteren rechten Ecke des Rechtecks.

Y2: Bestimmt die virtuelle y-Koordinate der unteren rechten Ecke des Rechtecks.

X3: Bestimmt die Breite der Ellipse, die benutzt wird, um die abgerundeten Ecken zu zeichnen.

Y3: Bestimmt die Höhe der Ellipse, die benutzt wird, um die abgerundeten Ecken zu zeichnen.

Rückgabewert Der Rückgabewert ist *True*, wenn das Rechteck gezeichnet wurde, andernfalls ist er *False*.

Hinweis Die Breite des Rechtecks, die sich aus der Differenz *X2 - X1* ergibt, darf 32767 Einheiten nicht überschreiten. Dieser Grenzwert gilt auch für die Höhe des Rechtecks (*Y2 - Y1*).

Die aktuelle Position wird von dieser Funktion weder benutzt noch aktualisiert.

SaveDC

Syntax
```
function SaveDC(DC: HDC): Integer;
```

Beschreibung Diese Funktion speichert den durch den Parameter *DC* bezeichneten aktuellen Status des Gerätekontexts, indem sie Statusinformationen (zum Beispiel die Clipping-Region, ausgewählte Objekte und den Abbildungsmodus) auf einen Kontext-Stack kopiert.

Der gespeicherte Gerätekontext kann später mit *RestoreDC* wiederhergestellt werden.

DC: Bezeichnet den Gerätekontext, der gespeichert werden soll.

Rückgabewert Der Rückgabewert gibt bei erfolgreicher Ausführung den gespeicherten Gerätekontext an, andernfalls ist er 0.

Hinweis Die Funktion *SaveDC* kann beliebig oft benutzt werden, um den Status beliebig vieler Gerätekontexte zu speichern.

Querverweis *RestoreDC*

ScaleViewportExt

Syntax
```
function ScaleViewportExt(DC: HDC; Xnum, Xdenom, Ynum,
Ydenom: Integer): LongInt;
```

Beschreibung Diese Funktion verändert die Abmessungen des Grafikfensters relativ zu den aktuellen Werten.

Dabei gilt folgende Formel:

xNeuVE = (xAltVE * *Xnum*) / *Xdenom*
yNeuVE = (yAltVE * *Ynum*) / *Ydenom*

Die neuen Abmessungen werden berechnet, indem die aktuellen Abmessungen erst mit dem angegebenen Zähler multipliziert und dann durch den angegebenen Nenner dividiert werden.

DC: Bezeichnet den Gerätekontext.

Xnum: Bestimmt den Wert, mit dem die aktuelle x-Abmessung multipliziert werden soll.

Xdenom: Bestimmt den Wert, durch den die aktuelle x-Abmessung dividiert werden soll.

Ynum: Bestimmt den Wert, mit dem die aktuelle y-Abmessung multipliziert werden soll.

Ydenom: Bestimmt den Wert, durch den die aktuelle y-Abmessung dividiert werden soll.

Rückgabewert Der Rückgabewert gibt die vorherigen Grafikfensterabmessungen in Geräteeinheiten an.

Die vorherige y-Abmessung liegt im höherwertigen, die vorherige x-Abmessung im niederwertigen Word.

ScaleWindowExt

Syntax
```
function ScaleWindowExt(DC: HDC; Xnum, Xdenom, Ynum,
Ydenom: Integer): LongInt;
```

Beschreibung Diese Funktion verändert die Abmessungen des Fensters relativ zu den aktuellen Werten.

Dabei gilt folgende Formel:

xNeuVE = (xAltVE * *Xnum*) / *Xdenom*
yNeuVE = (yAltVE * *Ynum*) / *Ydenom*

Die neuen Abmessungen werden berechnet, indem die aktuellen Abmessungen erst mit dem angegebenen Zähler multipliziert und dann durch den angegebenen Nenner dividiert werden.

DC: Bezeichnet den Gerätekontext

Xnum: Bestimmt den Wert, mit dem die aktuelle x-Abmessung multipliziert werden soll.

Xdenom: Bestimmt den Wert, durch den die aktuelle x-Abmessung dividiert werden soll.

Ynum: Bestimmt den Wert, mit dem die aktuelle y-Abmessung multipliziert werden soll.

Ydenom: Bestimmt den Wert, durch den die aktuelle y-Abmessung dividiert werden soll.

Rückgabewert Der Rückgabewert gibt die vorherigen Fensterabmessungen in virtuellen Einheiten an. Die vorherige y-Abmessung liegt im höherwertigen, die vorherige x-Abmessung im niederwertigen Word.

ScreenToClient

Syntax
```
procedure ScreenToClient(Wnd: HWnd; var Point: TPoint);
```

Beschreibung Diese Prozedur konvertiert die Bildschirmkoordinaten eines gegebenen Punktes auf dem Bildschirm in Client-Koordinaten.

ScreenToClient benutzt das Fenster, das durch den Parameter *Wnd* bezeichnet ist und die Bildschirmkoordinaten, die in der *TPoint*-Datenstruktur *Point* angegeben sind, um die Client-Koordinaten zu berechnen. Sie ersetzt dann die Bildschirmkoordinaten durch die Client-Koordinaten. Die neuen Koordinaten gelten relativ zur linken oberen Ecke des Client-Bereichs des angegebenen Fensters.

Wnd: Bezeichnet das Fenster, dessen Client-Bereich für die Konvertierung benutzt wird.

Point: Eine *TPoint*-Datenstruktur, die die zu konvertierenden Bildschirmkoordinaten enthält.

Hinweis Die Formel in *ScreenToClient* setzt voraus, daß der Punkt in Bildschirmkoordinaten angegeben ist.

ScrollDC

Syntax
```
function ScrollDC(DC: HDC; dx, dy: Integer; var Scroll,
Clip: TRect; UpdateRgn: HRgn; UpdateRect: PRect): Bool;
```

Beschreibung Diese Funktion rollt ein Rechteck aus Bits in horizontaler und vertikaler Richtung.

Der Parameter *Scroll* enthält das Rechteck, das gerollt werden soll. Der Parameter *dx* bezeichnet die Zahl der Einheiten, um die horizontal, *dy* die Zahl der Einheiten, um die vertikal gerollt werden soll.

DC: Bezeichnet den Gerätekontext, der die Bits enthält, die gerollt werden sollen.

dx: Bestimmt die Zahl der horizontalen Bildlaufeinheiten.

dy: Bestimmt die Zahl der vertikalen Bildlaufeinheiten.

Scroll: Die *TRect*-Datenstruktur mit den Koordinaten des zu rollenden Rechtecks.

Clip: Die *TRect*-Datenstruktur mit den Koordinaten des Clipping-Rechtecks. Wenn dieses Rechteck kleiner ist als das Original in *Scroll*, findet der Bildlauf nur in dem kleineren Rechteck statt.

UpdateRgn: Zeigt auf die Region, die vom Bildlauf beeinflußt wird. *ScrollDC* definiert diese Region, die nicht unbedingt ein Rechteck ist.

UpdateRect: Zeigt auf die *TRect*-Datenstruktur, die bei der Rückkehr die Koordinaten des Rechtecks enthält, das den Bildlaufbereich umfaßt. Dies ist der größte rechteckige Bereich, der ein Neuzeichnen erfordert.

Rückgabewert Der Rückgabewert ist *True*, wenn der Bildlauf ausgeführt wurde, andernfalls ist er *False*.

Hinweis Windows berechnet das Aktualisierungsrechteck nicht, wenn der Parameter *UpdateRect* 0 ist, die Aktualisierungsregion wird nicht berechnet, wenn sowohl *UpdateRect* als auch *UpdateRgn* 0 sind. Ist *UpdateRgn* nicht 0, nimmt Windows an, daß er ein gültiges Region-Handle zu der Region enthält, die vom Bildlauf betroffen wird (definiert durch die Funktion *ScrollDC*).

Wenn es notwendig ist, den gesamten Client-Bereich eines Fensters zu rollen, sollte eine Anwendung die Prozedur *ScrollWindow* benutzen, andernfalls sollte sie *ScrollDC* benutzen.

ScrollWindow

Syntax
```
procedure ScrollWindow(Wnd: HWnd; XAmount, YAmount:
Integer; Rect, ClipRect: PRect);
```

Beschreibung Diese Prozedur verschiebt ein Fenster, indem sie die Inhalte des Client-Bereichs des Fensters um die durch den Parameter *XAmount* vorgegebene Anzahl von Einheiten entlang der x-Achse und um die durch den Parameter *YAmount* vorgegebene Anzahl von Einheiten entlang der y-Achse des Bildschirms verschiebt.

Der Bildlauf erfolgt nach rechts, wenn *XAmount* positiv ist und nach links, wenn dieser Parameter negativ ist. Der Bildlauf erfolgt nach unten, wenn der Parameter *YAmount* positiv ist und nach oben, wenn er negativ ist.

Wnd: Bezeichnet das Fenster, für dessen Client-Bereich ein Bildlauf durchgeführt werden soll.

XAmount: Bestimmt den Betrag in gerätebezogenen Einheiten, um den in x-Richtung gerollt werden soll.

YAmount: Bestimmt den Betrag in gerätebezogenen Einheiten, um den in y-Richtung gerollt werden soll.

Rect: Zeigt auf eine *TRect*-Datenstruktur, die den Teil des Client-Bereiches bezeichnet, der gerollt werden soll. Ist *Rect* 0, wird der gesamte Client-Bereich gerollt.

ClipRect: Zeigt auf eine *TRect*-Datenstruktur, die das Clipping-Rechteck bezeichnet, das gerollt werden soll. Nur Bits innerhalb des Rechtecks werden gerollt. Ist *ClipRect* 0, findet der Bildlauf im gesamten Fenster statt.

Hinweis

Befindet sich das Caret in dem Fenster, das gerade gerollt wird, blendet *ScrollWindow* das Caret automatisch aus, um zu verhindern, daß es gelöscht wird. Nach Beendigung des Bildlaufs wird das Caret wieder hergestellt, die Position des Caret wird entsprechend angeglichen.

Der von *ScrollWindow* nicht abgedeckte Bereich wird nicht neu gezeichnet, sondern mit der Aktualisierungsregion des Fensters kombiniert. Die Anwendung empfängt eventuell eine Botschaft *wm_Paint,* die sie davon in Kenntnis setzt, daß die Region neu gezeichnet werden muß. Um den nicht abgedeckten Bereich zur selben Zeit neu zu zeichnen, zu der der Bildlauf durchgeführt wird, sollte *UpdateWindow* sofort nach dem Aufruf von *ScrollWindow* aufgerufen werden. Wenn der Parameter *Rect* 0 ist, werden die Positionen aller untergeordneter Fenster im Fenster um den Betrag verschoben, den *XAmount* und *YAmount* vorgeben; sämtliche unvollständigen (nicht gezeichneten) Bereiche im Fenster werden ebenso verschoben. *ScrollWindow* ist schneller, wenn *Rect* 0 ist.

Ist der Parameter *Rect* nicht 0, werden die Positionen untergeordneter Fenster nicht verändert und ungültige Bereiche im Fenster werden nicht verschoben. Um hier Probleme bei der Aktualisierung zu vermeiden, sollte man *UpdateWindow* aufrufen, um das Fenster vor dem Aufruf von *ScrollWindow* neu zu zeichnen.

Querverweis

UpdateWindow, wm_Paint

SelectClipRgn

Syntax

```
function SelectClipRgn(DC: HDC; Rgn: HRgn): Integer;
```

Beschreibung

Diese Funktion wählt die angegebene Region als die aktuelle Clipping-Region für den bezeichneten Gerätekontext. Es wird

nur eine Kopie der ausgewählten Region benutzt. Die Region selbst kann für eine beliebige Anzahl von anderen Gerätekontexten ausgewählt oder gelöscht werden.

DC: Bezeichnet den Gerätekontext

Rgn: Bezeichnet die Region, die ausgewählt werden soll.

Rückgabewert Der Rückgabewert gibt den Typ der Region an. Er kann einen der folgenden Werte annehmen:

Wert	Bedeutung
ComplexRegion	Neue Clipping-Region hat überlappende Grenzen.
Error	Gerätekontext oder Region-Handle nicht gültig.
NullRegion	Neue Clipping-Region ist leer.
SimpleRegion	Neue Clipping-Region hat keine überlappenden Grenzen.

Hinweis *SelectClipRgn* geht davon aus, daß die Koordinaten der gegebenen Region in Geräteeinheiten vorgegeben sind.

Einige Druckgeräte unterstützen die Grafikdarstellung mit niedrigeren Auflösungen als Textdarstellung, um die Geschwindigkeit bei gleichzeitiger Qualitätseinbuße zu erhöhen. Diese Geräte skalieren Koordinaten für Grafiken so, daß ein Grafikpunkt auf dem Druckgerät zwei oder vier echten Gerätepunkten entspricht. Dieser Skalierungsfaktor beeinflußt das Clipping. Wenn eine Region benutzt werden soll, um Grafiken auszuschneiden, dann müssen seine Koordinaten durch den Skalierungsfaktor geteilt werden. Soll die Region benutzt werden, um Text auszuschneiden, ist keine Angleichung der Skalierung notwendig. Der Skalierungsfaktor wird durch die Druckersteuersequenz *GetScalingFactor* bestimmt.

SelectObject

Syntax
```
function SelectObject(DC: HDC; hObject: THandle):
THandle;
```

Beschreibung Diese Funktion wählt das virtuelle Objekt, das der Parameter *hObject* bezeichnet, als ausgewähltes Objekt des angegebenen Gerätekontexts. Das neue Objekt ersetzt das vorherige Objekt gleichen Typs. Ist zum Beispiel *hObject* das Handle zu einem virtuellen Stift, ersetzt die Funktion *SelectObject* den ausgewählten Stift durch den von *hObject* vorgegebenen Stift.

Ausgewählte Objekte sind die Standardobjekte, die von den GDI-Ausgabefunktionen benutzt werden, um Linien zu zeichnen, Innenflächen auszufüllen, Text zu schreiben und Ausgaben auf spezielle Bereiche der Bildschirmoberfläche zu tätigen. Obwohl ein Gerätekontext sechs ausgewählte Objekte

haben kann (Pinsel, Stift, Schrift, Bitmap, Region und virtuelle Palette), kann nicht mehr als ein Objekt des angegebenen Typs zur gleichen Zeit ausgewählt sein. *SelectObject* wählt keine virtuelle Palette aus, dazu muß *SelectPalette* aufgerufen werden.

DC: Bezeichnet den Gerätekontext.

hObject: Bezeichnet das Objekt, das ausgewählt werden soll. Es kann eines der folgenden sein und muß vorher mit einer der folgenden Funktionen erzeugt worden sein (Bitmaps können nur für Speichergerätekontexte und immer nur für einen Gerätekontext ausgewählt werden):

Objekt	Funktion
Bitmap	*CreateBitmap*
	CreateBitmapIndirect
	CreateCompatibleBitmap
	CreateDIBitmap
Pinsel	*CreateBrushIndirect*
	CreateHatchBrush
	CreatePatternBrush
	CreateSolidBrush
Schrift	*CreateFont*
	CreateFontIndirect
Stift	*CreatePen*
	CreatePenIndirect
Region	*CombineRgn*
	CreateEllipticRgn
	CreateEllipticRgnIndirect
	CreatePolygonRgn
	CreateRectRgn
	CreateRectRgnIndirect

Rückgabewert

Der Rückgabewert bezeichnet das Objekt, das durch das vom Parameter *hObject* vorgegebene Objekt ersetzt wurde. Er ist 0, wenn ein Fehler auftritt. Wenn der Parameter *DC* ein Metafile bezeichnet, ist der Rückgabewert bei erfolgreicher Ausführung der Funktion ungleich 0, andernfalls ist er 0.

Wenn eine Region ausgewählt ist, dann ist der Rückgabewert der gleiche wie für *SelectClipRgn*.

Hinweis

Wird eine Schrift, ein Stift oder ein Pinsel mit *SelectObject* ausgewählt, reserviert die GDI Speicher in seinem Datensegment für dieses Objekt. Da der Datensegmentspeicher begrenzt ist, sollte man die Funktion *DeleteObject* benutzen, um jedes zeichnende Objekt zu löschen, das man nicht mehr benötigt. Vor dem Löschen des letzten nicht benötigten zeichnenden Objekts sollte die Anwendung das ursprüngliche (Standard-) Objekt in den Gerätekontext zurück wählen.

Eine Anwendung kann ein Bitmap nicht in mehr als einen Gerätekontext auf einmal auswählen.

Querverweis *DeleteObject, SelectClipRgn, SelectPalette*

SelectPalette

Syntax
```
function SelectPalette(DC: HDC; Palette: HPalette;
ForceBackground: Bool): HPalette;
```

Beschreibung Diese Funktion wählt die durch den Parameter *Palette* bezeichnete virtuelle Palette als ausgewähltes Palettenobjekt des durch den Parameter *DC* bezeichneten Gerätekontextes aus. Die neue Palette wird das von der GDI bei der Steuerung der im Gerätekontext dargestellten Farben benutzte Palettenobjekt und ersetzt die vorherige Palette.

DC: Bezeichnet den Gerätekontext.

Palette: Bezeichnet die virtuelle Palette, die ausgewählt werden soll. *CreatePalette* erstellt eine virtuelle Palette.

ForceBackground: Bestimmt, ob die virtuelle Palette eine Hintergrundpalette sein soll. Ist er *True,* ist die ausgewählte Palette immer eine Hintergrundpalette, ungeachtet dessen, ob das Fenster den Fokus hat. Ist er *False,* ist die virtuelle Palette eine Vordergrundpalette, wenn das Fenster den Fokus hat.

Rückgabewert Der Rückgabewert bezeichnet die virtuelle Palette, die von der durch *Palette* vorgegebenen Palette ersetzt wurde. Er ist 0, wenn ein Fehler vorliegt.

Hinweis Eine Anwendung kann eine virtuelle Palette in mehr als einem Gerätekontext auswählen. Änderungen an einer virtuellen Palette beeinflussen jedoch alle Gerätekontexte, für die sie ausgewählt ist. Wählt eine Anwendung ein Palettenobjekt für mehr als einen Gerätekontext aus, müssen die Gerätekontexte alle zu demselben physikalischen Gerät gehören (wie etwa einem Bildschirm oder einem Drucker).

Querverweis *CreatePalette*

SendDlgItemMessage

Syntax
```
function SendDlgItemMessage(Dlg: HWnd; IDDlgItem:
Integer; Msg, wParam: Word; lParam: LongInt): LongInt;
```

Beschreibung Diese Funktion sendet eine Botschaft an ein durch den Parameter *IDDlgItem* bezeichnetes Steuerelement innerhalb des durch den Parameter *Dlg* bezeichneten Dialogfensters. Die Funktion *SendDlgItemMessage* kehrt nicht zurück, bis die Botschaft verarbeitet wurde.

Dlg: Bezeichnet das Dialogfenster, in dem sich das Steuerelement befindet.

IDDlgItem: Bestimmt den Integerbezeichner des Dialogelements, das die Botschaft empfangen soll.

Msg: Bestimmt den Botschaftswert.

wParam: Bestimmt zusätzliche Botschaftsinformationen.

lParam: Bestimmt zusätzliche Botschaftsinformationen.

Rückgabewert Der Rückgabewert ist der Wert, der von der Fensterfunktion des Steuerelements zurückgegeben wurde oder 0, wenn der Bezeichner des Steuerelements nicht gültig ist.

Hinweis Die Verwendung von *SendDlgItemMessage* ist identisch mit dem Abruf des Handle zu dem angegebenen Steuerelement mit anschließendem Aufruf der Funktion *SendMessage*.

SendMessage

Syntax
```
function SendMessage(Wnd: HWnd; Msg, wParam: Word;
lParam: LongInt): LongInt;
```

Beschreibung Diese Funktion sendet eine Botschaft an ein oder mehrere Fenster. *SendMessage* kehrt solange nicht zurück, bis die Botschaft verarbeitet wurde. Ist das Fenster, das die Botschaft empfängt, Bestandteil der gleichen Anwendung, wird die Fensterfunktion sofort als eine Subroutine aufgerufen. Ist das Fenster Bestandteil einer anderen Task, schaltet Windows zu der richtigen Task, ruft die geeignete Fensterfunktion auf und übergibt dann die Botschaft an die Fensterfunktion. Die Botschaft wird nicht in der Warteschlange der Zielanwendung plaziert.

Wnd: Bezeichnet das Fenster, das die Botschaft empfangen soll. Ist der Parameter *Wnd* $FFFF, wird die Botschaft an alle Popup-Fenster im System übergeben. Die Botschaft wird nicht an untergeordnete Fenster gesendet.

Msg: Bestimmt die zu sendende Botschaft.

wParam: Bestimmt zusätzliche Botschaftsinformationen.

lParam: Bestimmt zusätzliche Botschaftsinformationen.

Rückgabewert Der Rückgabewert ist der Wert, der von der Fensterfunktion, die die Botschaft empfangen hat, zurückgegeben wird: Sein Wert hängt von der gesendeten Botschaft ab.

Hinweis Wenn ein System, in dem Windows läuft, für Expanded Memory (EMS) konfiguriert ist und eine Anwendung mit der Funktion *SendMessage* eine Botschaft mit verknüpften Daten

(auf die der Parameter *lParam* zeigt) an eine zweite Anwendung sendet, muß die erste Anwendung diese Daten in dem globalen Speicherbereich plazieren, der von der Funktion *GlobalAlloc* mit dem Flag *gmem_Lower* reserviert wurde. Diese Speicherreservierung ist nur notwendig, wenn *lParam* einen Zeiger enthält.

SetActiveWindow

Syntax
```
function SetActiveWindow(Wnd: HWnd): HWnd;
```

Beschreibung Diese Funktion macht ein Hauptfenster zum aktiven Fenster.

Wnd: Bezeichnet das zu aktivierende Hauptfenster.

Rückgabewert Der Rückgabewert bezeichnet das Fenster, das vorher aktiv war. Die Funktion *SetActiveWindow* sollte mit Vorsicht angewandt werden, da sie einer Anwendung erlaubt, das aktive Fenster und den Fokus unumschränkt zu übernehmen. Normalerweise überwacht Windows jede Aktivierung.

SetBitmapBits

Syntax
```
function SetBitmapBits(Bitmap: HBitmap; Count: LongInt;
Bits: Pointer): LongInt;
```

Beschreibung Diese Funktion setzt die Bits eines Bitmaps auf die Werte, die der Parameter *Bits* vorgibt.

Bitmap: Bezeichnet das Bitmap, das gesetzt werden soll.

Count: Bestimmt die Zahl der Bytes, auf die *Bits* zeigt.

Bits: Zeigt auf die Bitmap-Bits, die in einem Byte-Array gespeichert sind.

Rückgabewert Der Rückgabewert gibt bei erfolgreicher Ausführung die Zahl der Bytes an, die beim Setzen der Bitmap-Bits benutzt wurden, andernfalls 0.

SetBitmapDimension

Syntax
```
function SetBitmapDimension(Bitmap, X, Y: Integer):
LongInt;
```

Beschreibung Diese Funktion weist einem Bitmap Breite und Höhe in Einheiten von 0.1 Millimeter zu. Diese Werte werden von der GDI intern nicht genutzt; die Funktion *GetBitmapDimension* kann benutzt werden, um sie abzurufen.

Bitmap: Bezeichnet das Bitmap.

X: Bestimmt die Breite des Bitmaps in Einheiten von 0.1 Millimeter.

Y: Bestimmt die Höhe des Bitmaps in Einheiten von 0.1 Millimeter.

Rückgabewert Der Rückgabewert gibt die vorherigen Abmessungen des Bitmaps an. Die Höhe liegt im höherwertigen, die Breite im niederwertigen Word.

SetBkColor

Syntax
```
function SetBkColor(DC: HDC; Color: TColorRef): TColorRef;
```

Beschreibung Diese Funktion setzt die aktuelle Hintergrundfarbe auf die durch den Parameter *Color* vorgegebene Farbe oder aber auf die nächste physikalische Farbe, wenn das Gerät diesen RGB-Farbwert nicht darstellen kann.

Wenn der Hintergrundmodus *Opaque* ist, benutzt die GDI die Hintergrundfarbe, um Lücken zwischen Linien bestimmter Arten, Lücken zwischen schraffierten Linien in Pinseln und Zeichenzellen auszufüllen. Die GDI benutzt die Hintergrundfarbe auch dazu, Bitmaps aus der Farbdarstellung in einfarbige Darstellung und umgekehrt zu konvertieren.

Der Hintergrundmodus wird durch die Funktion *SetBkMode* gesetzt. Siehe auch *BitBlt* und *StretchBlt*.

DC: Bezeichnet den Gerätekontext.

Color: Bestimmt die neue Hintergrundfarbe.

Rückgabewert Der Rückgabewert gibt die vorherige Hintergrundfarbe als einen RGB-Farbwert an, tritt ein Fehler auf, ist der Rückgabewert $80000000.

Querverweis *BitBlt, SetBkMode, StretchBlt*

SetBkMode

Syntax
```
function SetBkMode(DC: HDC; BkMode: Integer): Integer;
```

Beschreibung Diese Funktion setzt den Hintergrundmodus, der für Text und Linien benutzt wird. Der Hintergrundmodus bestimmt, ob die GDI vorhandene Hintergrundfarben auf der Bildschirmfläche löschen soll, bevor Text, Pinselschraffur oder irgendein Stift, der keine durchgezogene Linie erzeugt, dargestellt wird.

DC: Bezeichnet den Gerätekontext.

BkMode: Bestimmt den Hintergrundmodus. Er kann einen der folgenden Werte annehmen:

Wert	Bedeutung
Opaque	Hintergrund wird mit der aktuellen Hintergrundfarbe ausgefüllt, bevor gezeichnet wird.
Transparent	Hintergrund bleibt unberührt.

Rückgabewert Der Rückgabewert gibt den vorherigen Hintergrundmodus an.

SetBrushOrg

Syntax
```
function SetBrushOrg(DC: HDC; X, Y: Integer): LongInt;
```

Beschreibung Diese Funktion setzt den Ursprung des aktuell ausgewählten Pinsels im gegebenen Gerätekontext.

DC: Bezeichnet den Geräteekontext.

X: Bestimmt die x-Koordinate des neuen Ursprungs in gerätespezifischen Einheiten. Der Wert muß im Bereich von 0 bis 7 liegen.

Y: Bestimmt die y-Koordinate des neuen Ursprungs in gerätespezifischen Einheiten. Der Wert muß im Bereich von 0 bis 7 liegen.

Rückgabewert Der Rückgabewert bestimmt den vorherigen Ursprung des Pinsels. Die vorherige x-Koordinate liegt im niederwertigen, die y-Koordinate im höherwertigen Word.

Hinweis Die Ausgangskoordinaten des Pinselursprungs sind (0,0). Die Funktion *SetBrushOrg* sollte nicht für vordefinierte Objekte benutzt werden.

SetCapture

Syntax
```
function SetCapture(Wnd: HWnd): HWnd;
```

Beschreibung Diese Funktion bewirkt, daß alle nachfolgenden Mauseingaben an das durch den Parameter *Wnd* bezeichnete Fenster gesendet werden, unabhängig von der Mauszeigerposition.

Wnd: Bezeichnet das Fenster, das alle Mauseingaben auf sich ziehen soll.

Rückgabewert Der Rückgabewert gibt das Fenster an, das vorher alle Mauseingaben erhalten hatte. Er ist 0, wenn keines existiert.

Hinweis Wenn das Fenster keine exkluxive Benutzung der Maus mehr benötigt, sollte man die Prozedur *ReleaseCapture* aufrufen, damit andere Fenster die Maus exklusiv benutzen können.

Querverweis *ReleaseCapture*

SetCaretBlinkTime

Syntax
```
procedure SetCaretBlinkTime(MSeconds: Word);
```

Beschreibung Diese Prozedur setzt die Blinkrate des Carets (die abgelaufene Zeit zwischen dem Aufblinken des Carets) auf die durch den Parameter *MSeconds* angegebene Zahl von Millisekunden.

Das Caret blinkt auf oder erlischt alle *MSeconds* Millisekunden. Das bedeutet, daß ein kompletter Blinkvorgang (AN-AUS-AN) 2 * *MSeconds* Millisekunden beansprucht.

MSeconds: Bestimmt die neue Blinkrate in Millisekunden.

Hinweis Das Caret ist eine gemeinsam genutzte Ressource. Ein Fenster sollte die Blinkrate des Carets nur setzen, wenn es das Caret besitzt, wenn es den Fokus verliert oder inaktiv wird, sollte es das vorherige Caret wiederherstellen.

SetCaretPos

Syntax
```
procedure SetCaretPos(X, Y: Integer);
```

Beschreibung Diese Prozedur bewegt das Caret zu der Position, die in virtuellen Koordinaten durch die beiden Parameter *X* und *Y* angegeben ist.

Virtuelle Kordinaten sind relativ zum Client-Bereich des Fensters gebildet, das sie besitzt, und werden vom Abbildungsmodus des Fensters beeinflußt, so daß die exakte Position in Pixeln von diesem Modus abhängt.

SetCaretPos bewegt das Caret nur, wenn es von einem Fenster der aktuellen Task besessen wird, unabhängig davon, ob das Caret ausgeblendet ist oder nicht.

X: Bestimmt die neue x-Koordinate des Carets in virtuellen Koordinaten.

Y: Bestimmt die neue y-Koordinate des Carets in virtuellen Koordinaten.

Hinweis Das Caret ist eine gemeinsam genutzte Ressource. Ein Fenster sollte das Caret jeweils nur dann bewegen, wenn es das Caret besitzt.

SetClassLong

Syntax
```
function SetClassLong(Wnd: HWnd; Index: Integer;
NewLong: LongInt): LongInt;
```

Beschreibung Diese Funktion ersetzt den vom Parameter *Index* bezeichneten LongInt in der *TWndClass*-Datenstruktur des durch den Parameter *Wnd* bezeichneten Fensters.

Wnd: Bezeichnet das Fenster.

Index: Bestimmt den Byte-Offset des LongInt, das ersetzt werden soll. Dieser Parameter kann auch einen der folgenden Werte annehmen:

Wert	Bedeutung
gcl_MenuName	Setzt einen neuen Zeiger auf die Menübezeichnung.
gcl_WndProc	Setzt einen neuen Zeiger auf die Fensterfunktion.

NewLong: Bestimmt den Ersatzwert.

Rückgabewert Der Rückgabewert zeigt den vorherigen Wert des angegebenen LongInt an.

Hinweis Wenn die Funktion *SetClassLong* und der Index *gcl_WndProc* benutzt werden, um eine Fensterfunktion zu setzen, muß die angegebene Funktion Fensterfunktionsform haben und in der Moduldefinitionsdatei exportiert werden.

Beachten Sie hierzu die Beschreibung Funktion *RegisterClass*.

Der Aufruf von *SetClassLong* mit dem Index *gcl_WndProc* erzeugt eine neue Unterklasse der Fensterklasse, die alle später mit dieser Klasse erzeugten Fenster betrifft. Eine Anwendung sollte keine Fensterunterklassen für Standard-Windows-Steuerelemente wie Kombinationsfenster und Schalter erstellen. Um auf zusätzliche Zwei-Byte-Werte zuzugreifen, die belegt wurden, als die Fensterklasse erzeugt wurde, benutzt man einen positiven Byte-Offset ab dem vom Parameter *nIndex* bezeichneten Index.

Dieser Index beginnt bei 0 für den ersten Zwei-Byte-Wert im zusätzlichen Speicherplatz, 2 für den nächsten Zwei-Byte-Wert und so weiter.

SetClassWord

Syntax
```
function SetClassWord(Wnd: HWnd; Index: Integer;
NewWord: Word): Word;
```

Beschreibung Diese Funktion ersetzt das vom Parameter *Index* bezeichnete Word in der *TWndClass*-Datenstruktur des durch den Parameter *Wnd* bezeichneten Fensters.

Wnd: Bezeichnet das Fenster.

Index: Bestimmt den Byte-Offset des Word, das verändert werden soll. Dieser Parameter kann einen der folgenden Werte annehmen:

Wert	Bedeutung
gcw_CBClsExtra	Setzt zwei neue Bytes zusätzlicher Fensterklassendaten.
gcw_CBWndExtra	Setzt zwei neue Bytes zusätzlicher Fensterklassendaten.
gcw_HBrBackground	Setzt ein neues Handle auf einen Hintergrundpinsel.
gcw_HCursor	Setzt ein neues Handle auf einen Mauszeiger.
gcw_HIcon	Setzt ein neues Handle auf ein Symbol.
gcw_Style	Setzt ein neues Stilbit für die Fensterklasse.

NewWord: Bestimmt den Ersatzwert.

Rückgabewert Der Rückgabewert gibt den vorherigen Wert des bezeichneten Word an.

Hinweis Die Funktion *SetClassWord* sollte mit Vorsicht angewandt werden. Zum Beispiel ist es möglich, die Hintergrundfarbe für eine Klasse mit *SetClassWord* zu ändern. Diese Änderung bewirkt aber nicht, daß alle Fenster, die dieser Klasse angehören, sofort neu gezeichnet werden.

Um auf zusätzliche Vier-Byte-Werte zuzugreifen, die belegt wurden, als die Fensterklasse erzeugt wurde, benutzt man einen positiven Byte-Offset ab dem vom Parameter *nIndex* bezeichneten Index. Dieser Index beginnt bei 0 für den ersten Vier-Byte-Wert im zusätzlichen Speicherplatz, 4 für den nächsten Vier-Byte-Wert und so weiter.

SetClipboardData

Syntax
```
function SetClipboardData(Format: Word; Mem: THandle):
THandle;
```

Beschreibung Diese Funktion setzt ein Daten-Handle zur Zwischenablage für die die durch den Parameter *Mem* bezeichneten Daten. Es wird vorausgesetzt, daß die Daten das vom Parameter *Format* vorgegebene Format haben.

Nach dem Setzen eines Zwischenablagedaten-Handles gibt die Funktion *SetClipboardData* den von *Mem* bezeichneten Speicherblock frei.

Format: Bestimmt ein Datenformat. Es kann eines der im Hinweis beschriebenen Formate sein. Zusätzlich zu den vordefinierten Formaten kann jeder mit der Funktion *RegisterClipboardFormat* registrierte Formatwert als Parameter *Format* benutzt werden.

Mem: Bezeichnet den globalen Speicherblock, der die Daten im angegebenen Format enthält. Der Parameter *Mem* kann auch

0 sein, in diesem Fall muß die Anwendung die Daten nicht formatieren und kein Handle auf sie bereitstellen, solange sie nicht durch eine Botschaft *wm_RenderFormat* dazu aufgefordert wird.

Rückgabewert Der Rückgabewert bezeichnet die Daten und wird von der Zwischenablage zugewiesen.

Hinweis Sobald der Parameter *Mem* an *SetClipboardData* übergeben wurde, wird der Datenblock Eigentum der Zwischenablage. Die Anwendung kann die Daten zwar lesen, sollte aber den Block nicht freigeben oder gesperrt lassen.

Windows unterstützt zwei Formate für Text, *cf_Text* und *cf_OEMText*. *cf_Text* ist das voreingestellte Zwischenablageformat für Text von Windows, das Format *cf_OEMText* wird für Text in Nicht-Windows-Anwendungen benutzt.

Wird *GetClipboardData* aufgerufen, um Daten in einem Textformat abzurufen und das andere Textformat ist dabei das einzig verfügbare, konvertiert Windows den Text automatisch in das erforderliche Format, bevor er der Anwendung übergeben wird.

Eine Anwendung registriert andere Standardformate, wie Rich Text Format (RTF), durch Benennung über die Funktion *RegisterClipboardFormat* anstatt mit einer symbolischen Konstante.

Wert	Bedeutung
cf_Bitmap	Ein Handle zu einem Bitmap (*HBitmap*).
cf_DIB	Ein Speicherblock, der die *TBitmapInfo*-Datenstruktur gefolgt von den Bitmap-Bits enthält.
cf_DIF	Software Arts' Data Interchange Format.
cf_DSPBitmap	Bitmap-Anzeigeformat, das einem privaten Format zugeordnet ist. Der Parameter *Mem* muß ein Handle zu Daten sein, die im Bitmap-Format anstelle eines privaten Formats dargestellt werden können.
cf_DSPMetafilePict	Metafile-Darstellungsformat, das einem privaten Format zugeordnet ist. Der Parameter *Mem* muß ein Handle zu Daten sein, die im Metafile-Bildformat statt eines privaten Formats dargestellt werden können.
cf_DSPText	Textdarstellungsformat, das einem privaten Format zugeordnet ist. Der Parameter *Mem* muß ein Handle zu Daten sein, die im Textformat anstelle eines privaten Formats dargestellt werden können.
cf_MetafilePict	Metafile-Bildformat wie von der *MetafilePict*-Datenstruktur definiert.

Wert	Bedeutung
cf_OEMText	Textformat, das Zeichen aus dem OEM-Zeichensatz enthält. Jede Zeile endet mit einer Kombination von Wagenrücklauf/Zeilenschaltung (CR-LF). Ein Null-Zeichen kennzeichnet das Datenende.
cf_OwnerDisplay	Besitzeranzeigeformat. Der Besitzer der Zwischenablage muß das Anwendungsfenster der Zwischenablage anzeigen und aktualisieren und empfängt die Botschaften *wm_ASKCBFormatName, wm_HScrollClipboard, wm_PaintClipboard, wm_SizeClipboard* und *wm_VScrollClipboard*. Der Parameter *Mem* muß 0 sein.
cf_Palette	Handle auf eine Farbpalette. Wenn eine Anwendung Daten in der Zwischenablage speichert, die von einer Farbpalette abhängen oder eine solche voraussetzen, sollte sie die Palette ebenfalls in der Zwischenablage speichern. Wenn die Zwischenablage Daten im Format *cf_Palette* (virtuelle Farbpalette) enthält, sollte die Anwendung annehmen, daß andere Daten in der Zwischenablage entgegen dieser virtuellen Palette realisiert sind. Die Zwischenablage-Viewer-Anwendung (CLIPBRD.EXE) benutzt als aktuelle Palette ein Objekt im Format *cf_Palette*, das in der Zwischenablage ist, wenn die anderen Formate in der Zwischenablage dargestellt werden.
cf_PrivateFirst bis *cf_Privatelast*	Bereich von Integerwerten, die für private Formate benutzt werden. Daten-Handles, die Formaten in diesem Bereich zugeordnet sind, werden nicht automatisch freigegeben. Alle Daten-Handles müssen freigegeben werden, bevor die Anwendung beendet oder wenn eine Botschaft *wm_DestroyClipboard* empfangen wird.
cf_Sylk	Microsoft-Symbolic-Link (SYLK) Format.
cf_Text	Textformat. Jede Zeile endet mit einer Kombination Wagenrücklauf/Zeilenschaltung. Ein Null-Zeichen kennzeichnet das Datenende.
cf_Tiff	Tag Image File Format.

SetClipboardViewer

Syntax
```
function SetClipboardViewer(Wnd: HWnd): HWnd;
```

Beschreibung Diese Funktion fügt das durch den Parameter *Wnd* bestimmte Fenster an die Kette von Fenstern an, die über die Botschaft

wm_DrawClipboard benachrichtigt werden, sobald der Inhalt der Zwischenablage geändert wird.

Wnd: Bezeichnet das Fenster, das Botschaften an die Zwischenablage-Viewer-Kette empfangen soll.

Rückgabewert Der Rückgabewert bezeichnet das nächste Fenster in der Kette der Zwischenablage-Viewer.

Dieses Handle sollte im statischen Speicher abgelegt werden und zur Beantwortung der Botschaften an die Kette der Zwischenablage-Viewer benutzt werden.

Hinweis Fenster, die Teil der Zwischenablage-Viewer-Kette sind, müssen auf die Botschaften *wm_ChangeCBChain, wm_DrawClipboard* und *wm_Destroy* antworten.

Wenn eine Anwendung sich selbst aus der Zwischenablage-Viewer-Kette löschen will, muß sie die Funktion *ChangeClipboardChain* aufrufen.

SetCommBreak

Syntax
```
function SetCommBreak(Cid: Integer): Integer;
```

Beschreibung Diese Funktion unterbricht die Zeichenübertragung und versetzt die Übertragungsleitung in einen Wartezustand, bis die Funktion *ClearCommBreak* aufgerufen wird.

Cid: Bestimmt die Schnittstelle, die unterbrochen werden soll. Die Funktion *OpenComm* gibt diesen Wert zurück.

Rückgabewert Der Rückgabewert ist 0, wenn die Funktion erfolgreich war und negativ, wenn *Cid* keine gültige Schnittstelle angibt.

Querverweis *ClearCommBreak, OpenComm*

SetCommEventMask

Syntax
```
function SetCommEventMask(Cid: Integer; EvtMask: Word):
PWord;
```

Beschreibung Diese Funktion aktiviert die Ereignismaske der vom Parameter *Cid* angegebenen Schnittstelle und ruft sie ab.

Die Bits des Parameters *EvtMask* definieren, welche Ereignisse aktiviert werden sollen. Der Rückgabewert zeigt auf den aktuellen Status der Ereignismaske.

Cid: Bestimmt die Schnittstelle, die aktiviert werden soll. Die Funktion *OpenComm* gibt diesen Wert zurück.

EvtMask: Bestimmt, welche Ereignisse aktiviert werden sollen. Er kann jede Kombination der im Hinweis aufgeführten Werte sein.

Rückgabewert Der Rückgabewert zeigt auf eine Word-Ereignismaske. Jedes Bit in der Ereignismaske gibt an, ob ein gegebenes Ereignis eingetreten ist, oder nicht. Ein Bit ist 1 (gesetzt), wenn das Ereignis eingetreten ist.

Folgende Ereignise sind möglich:

Wert	Bedeutung
ev_Break	Gesetzt, wenn eine Unterbrechung bei der Eingabe erkannt wurde.
ev_CTS	Gesetzt, wenn das Signal Clear-To-Send (CTS) den Status wechselt.
ev_DSR	Gesetzt, wenn das Signal Data-Set-Ready (DSR) den Status wechselt.
ev_Err	Gesetzt, wenn ein Leitungsstatusfehler auftritt. Leitungsstatusfehler sind *ce_Frame, ce_Overrun* und *ce_RXParity*.
ev_PErr	Gesetzt, wenn ein Druckerfehler an einer parallelen Schnittstelle erkannt wurde. Fehler sind *ce_DNS, ce_IOE, CE_Loop* und *ce_PTO*.
ev_Ring	Gesetzt, wenn ein Anrufsignal erkannt wurde.
ev_RLSD	Gesetzt, wenn das Signal Receive-Line-Signal-Detect (RLSD) den Status ändert.
ev_RxChar	Gesetzt, wenn ein Zeichen empfangen und in die Warteschlange gestellt wurde.
ev_RxFlag	Gesetzt, wenn das Ereigniszeichen empfangen und in die Empfangs-Warteschlange gestellt wurde. Das Ereigniszeichen wird im Gerätesteuerblock des Gerätes bestimmt.
ev_TxEmpty	Gesetzt, wenn das letzte Zeichen in der Sende-Warteschlange gesendet wurde.

SetCommState

Syntax
```
function SetCommState(var DCB: TDCB): Integer;
```

Beschreibung Diese Funktion versetzt eine Schnittstelle in den Status, den der Gerätesteuerblock in der *TDCB*-Struktur des Parameters *DCB* vorgibt. Das zu setzende Gerät muß durch das ID-Feld des Steuerblocks bezeichnet werden.

Diese Funktion initialisiert die gesamte Hardware und die Steuerleitungen gemäß der Definition in *DCB*. Sie leert keine Sende- oder Empfangs-Warteschlangen.

DCB: Eine *TDCB*-Datenstruktur, die die gewünschten Schnittstelleneinstellungen für das Gerät enthält.

Rückgabewert Der Rückgabewert ist 0, wenn die Funktion erfolgreich war. Er ist negativ, wenn ein Fehler auftritt.

SetCursor

Syntax
```
function SetCursor(Cursor: HCursor): HCursor;
```

Beschreibung Diese Funktion setzt die Mauszeigerform auf die durch den Parameter *Cursor* bezeichnete Form. Der Mauszeiger wird nur gesetzt, wenn sich die neue Form von der aktuellen Form unterscheidet, andernfalls kehrt die Funktion zurück. Die Funktion *SetCursor* ist sehr schnell, wenn der durch den Parameter *Cursor* bezeichnete Mauszeiger mit dem aktuellen Mauszeiger identisch ist.

Wenn *Cursor* 0 ist, wird der Mauszeiger vom Bildschirm gelöscht.

Cursor: Bezeichnet die Mauszeiger-Ressource. Die Ressource muß vorher mit der Funktion *LoadCursor* geladen worden sein.

Rückgabewert Der Rückgabewert bezeichnet die Mauszeiger-Ressource, die die vorherige Mauszeigerform definiert. Er ist 0, wenn es keine vorherige Mauszeigerform gibt.

Hinweis Der Mauszeiger ist eine gemeinsam genutzte Ressource. Eine Anwendung sollte den Mauszeiger nur setzen, wenn der Mauszeiger in ihrem Client-Bereich ist oder sie den Mausfang hat. In Systemen ohne Maus sollte das Fenster die vorherige Mauszeigerform wiederherstellen, bevor der Mauszeiger den Client-Bereich verläßt oder bevor das Fenster einem anderen Fenster die Steuerung überläßt. Jede Anwendung, die die Mauszeigerform ändern muß, während dieser in einem Fenster ist, muß sicherstellen, daß der Klassenzeiger für die angegebene Klasse des Fensters auf 0 gesetzt ist. Wenn der Klassenzeiger nicht 0 ist, stellt Windows jedesmal, wenn die Maus bewegt wird, die vorherige Form wieder her. Der Mauszeiger ist auf dem Bildschirm nicht sichtbar, wenn sein Anzeigezähler durch entsprechende Aufrufe von *ShowCursor* unter 0 gesunken ist.

SetCursorPos

Syntax
```
procedure SetCursorPos(X, Y: Integer);
```

Beschreibung Diese Prozedur bewegt den Mauszeiger zu den durch die Parameter *X* und *Y* angegebenen Bildschirmkoordinaten. Wenn die neuen Koordinaten nicht innerhalb des Bildschirmrechtecks liegen, das durch den letzten Aufruf der Prozedur *ClipCursor* gesetzt wurde, berichtigt Windows automatisch die Koordinaten so, daß der Mauszeiger innerhalb des Rechtecks bleibt.

X: Bestimmt die neue x-Koordinate des Mauszeigers in Bildschirmkoordinaten.

Y: Bestimmt die neue y-Koordinate des Mauszeigers in Bildschirmkoordinaten.

Hinweis

Der Mauszeiger ist eine gemeinsam genutzte Ressource. Ein Fenster sollte den Mauszeiger nur bewegen, wenn der Mauszeiger in seinem Client-Bereich liegt.

SetDIBits

Syntax

```
function SetDIBits(DC: HDC; Bitmap: THandle; StartScan,
NumScans: Word; Bits: Pointer; var BitsInfo:
TBitmapInfo; Usage: Word ): Integer;
```

Beschreibung

Diese Funktion setzt die Bits eines Bitmaps auf die in einer geräteunabhängigen Bitmapbeschreibung angegebenen Werte.

DC: Bezeichnet den Gerätekontext.

Bitmap: Bezeichnet das Bitmap.

StartScan: Bestimmt die Nummer der ersten Scan-Zeile in *Bits*.

NumScans: Bestimmt die Anzahl von Scan-Zeilen in *Bits* und die Anzahl von Zeilen, die in dem vom Parameter *Bitmap* bezeichneten Bitmap gesetzt werden sollen.

Bits: Zeigt auf die geräteunabhängigen Bitmapbits, die als ein Array von Bytes gespeichert sind. Das Format der Bitmap-Werte hängt vom Feld *biBitCount* der *TBitmapInfo*-Datenstruktur in *BitsInfo* ab.

BitsInfo: Eine *TBitmapInfo*-Datenstruktur, die Informationen über das geräteunabhängige Bitmap enthält.

Usage: Bestimmt, ob die Felder *bmiColors[ ]* in *BitsInfo* explizite RGB-Werte oder Indizes in die aktuell realisierte virtuelle Palette enthalten. Der Parameter *Usage* muß einen der folgenden Werte annehmen:

Wert	Bedeutung
DIB_Pal_Colors:	Die Farbtabelle besteht aus einem Array von 16-Bit-Indizes in die aktuell realisierte virtuelle Palette.
DIB_RGB_Colors:	Die Farbtabelle enthält explizite RGB-Werte.

Rückgabewert

Bei erfolgreicher Ausführung wird die Anzahl von Scan-Zeilen, die erfolgreich kopiert wurden, zurückgeliefert, andernfalls 0.

Hinweis

Das Bitmap, das vom Parameter *Bitmap* bezeichnet wird, darf nicht für einen Gerätekontext gewählt sein, wenn die Anwendung diese Funktion aufruft.

Der Ursprung für geräteunabhängige Bitmaps ist die untere linke Ecke des Bitmaps (die obere linke Ecke ist der Ursprung, wenn der Abbildungsmodus *mm_Text* ist).

SetDIBitsToDevice

Syntax

```
function SetDIBitsToDevice(DC: HDC; DestX, DestY,
Width, Height, SrcX, SrcY, StartScan, NumScans: Word;
Bits: Pointer; var BitsInfo: TBitmapInfo; Usage: Word):
Integer;
```

Beschreibung

Diese Funktion setzt Bits aus einem geräteunabhängigen Bitmap (DIB) direkt auf eine Bildschirmoberfläche. Die Parameter *SrcX*, *SrcY*, *Width* und *Height* definieren ein Rechteck innerhalb des gesamten DIB. *SetDIBitsToDevice* setzt die Bits in diesem Rechteck direkt auf der Bildschirmfläche des durch den Parameter *DC* bezeichneten Ausgabegerätes an die durch die Parameter *DestX* und *DestY* festgelegte Position.

Um den erforderlichen Speicherbedarf beim Setzen von Bits aus einem großen DIB auf eine Bildschirmfläche zu verringern, kann eine Anwendung die Ausgabe durch wiederholtes Aufrufen von *SetDIBitsToDevice* eingrenzen, indem sie dabei jedes Mal einen anderen Teil des gesamten DIB in *Bits* bearbeitet. Die Werte der Parameter *StartScan* und *NumScan* bezeichnen den Teil des gesamten DIB, der im Puffer *Bits* enthalten ist.

DC: Bezeichnet den Gerätekontext.

DestX: Bestimmt die x-Koordinate des Ursprungs des Zielrechtecks.

DestY: Bestimmt die y-Koordinate des Ursprungs des Zielrechtecks.

Width: Bestimmt die x-Abmessung des Rechtecks im DIB.

Height: Bestimmt die y-Abmessung des Rechtecks im DIB.

SrcX: Bestimmt die x-Koordinate der Quelle im DIB.

SrcY: Bestimmt die y-Koordinate der Quelle im DIB.

StartScan: Bestimmt die Nummern der Scan-Zeilen des DIB, die in der ersten Scan-Zeile des Puffers in *Bits* enthalten sind.

NumScans: Bestimmt die Anzahl von Scan-Zeilen des DIB, die in *Bits* enthalten sind.

Bits: Byte-Aray, das die Bits des geräteunabhängigen Bitmaps enthält. Das Format der Bits ist durch das Feld *BitsCount* in *BitsInfo* festgelegt.

BitsInfo: Eine *TBitmapInfo*-Datenstruktur, die Information über das DIB enthält.

Usage: Bestimmt, ob die Felder *bmiColors[ ]* in *BitsInfo* explizite RGB-Werte oder Indizes in die aktuell realisierte virtuelle Palette enthalten. Der Parameter *Usage* muß einer der folgenden Werte sein:

Wert	Bedeutung
DIB_Pal_Colors:	Die Farbpalette besteht aus einem Array von 16-Bit-Indizes in die aktuell realisierte virtuelle Palette.
DIB_RGB_Colors:	Die Farbtabelle enthält explizite RGB-Werte.

Rückgabewert Bei erfolgreicher Ausführung ist der Rückgabewert die Zahl der gesetzten Scan-Zeilen, andernfalls 0.

Hinweis Alle Koordinaten außer den virtuellen Koordinaten *DestX* und *DestY* sind gerätebezogene Koordinaten (also die Koordinaten des DIB).

Der Ursprung für geräteunabhängige Bitmaps ist die untere linke Ecke des DIB (die obere linke Ecke ist der Ursprung, wenn der Abbildungsmodus *mm_Text* ist).

SetDlgItemInt

Syntax
```
procedure SetDlgItemInt(Dlg: HWnd; IDDlgItem: Integer
Value: Word; Signed: Bool);
```

Beschreibung Diese Prozedur setzt den Text eines Steuerelements in dem angegebenen Dialogfenster auf den String, den der vom Parameter *Value* gegebene Integerwert repräsentiert. Die Prozedur *SetDlgItemInt* wandelt *Value* in einen aus Dezimalziffern bestehenden String um und kopiert den String dann in das Steuerelement. Wenn der Parameter *Signed True* ist, wird angenommen, daß *Value* vorzeichenbehaftet ist. Ist *Value* vorzeichenbehaftet und kleiner 0, setzt die Prozedur ein Minuszeichen vor die erste Ziffer im String.

SetDlgItemInt sendet eine Botschaft *wm_SetText* an das angegebene Steuerelement.

Dlg: Bezeichnet das Dialogfenster, das das Steuerelement enthält.

IDDlgItem: Bestimmt das Steuerelement, das verändert werden soll.

Value: Bestimmt den Wert, der gesetzt werden soll.

Signed: Bestimmt, ob der Wert vorzeichenbehaftet ist oder nicht.

Querverweis *wm_SetText*

SetDlgItemText

Syntax
```
procedure SetDlgItemText(Dlg: HWnd; IDDlgItem: Integer;
Str: PChar);
```

Beschreibung Diese Prozedur setzt die Überschrift oder den Text eines Steuerelements in das durch den Parameter *Dlg* bezeichnete Dialogfenster. Sie sendet eine Botschaft *wm_SetText* an das angegebene Steuerelement.

Dlg: Bezeichnet das Dialogfenster, welches das Steuerelement enthält.

IDDlgItem: Bestimmt das Steuerelement, dessen Text gesetzt werden soll.

Str: Zeigt auf einen nullterminierten String, der in das Steuerelement kopiert werden soll.

Querverweis *wm_SetText*

SetDoubleClickTime

Syntax
```
procedure SetDoubleClickTime(Count: Word);
```

Beschreibung Diese Prozedur setzt das Doppelklickzeitintervall für die Maus.

Ein Doppelklick ist eine Folge von zwei Klicks mit der Maustaste, wobei der zweite innerhalb einer festgelegten Zeitspanne nach dem ersten erfolgt. Das Doppelklickzeitintervall ist die maximale Zahl von Millisekunden, die zwischen dem ersten und zweiten Klick eines Doppelklicks verstreichen dürfen.

Count: Bestimmt die Zahl der Millisekunden, die zwischen den beiden Klicks liegen dürfen.

Hinweis Ist der Parameter *Count* auf 0 gesetzt, verwendet Windows das Standardzeitintervall von 500 Millisekunden für den Doppelklick.

Die Prozedur *SetDoubleClickTime* wechselt das Doppelklickzeitintervall für alle Fenster im System.

SetEnvironment

Syntax
```
function SetEnvironment(PortName: Char; Environ:
Pointer; Count: Word) Integer;
```

Beschreibung Diese Funktion kopiert den Inhalt des durch den Parameter *Environ* bezeichneten Puffers in die Umgebung, die dem Gerät zugeordnet ist, das an den vom Parameter *PortName* bezeichneten Anschluß angeschlossen ist. *SetEnvironment* löscht jede

existierende Umgebung. Gibt es keine Umgebung für den angegebenen Anschluß, wird diese von *SetEnvironment* erzeugt. Wenn der Parameter *Count* 0 ist, wird die vorhandene Umgebung nur gelöscht und nicht ersetzt.

PortName: Zeigt auf einen nullterminierten String, der die Bezeichnung des gewünschten Anschlusses enthält.

Environ: Zeigt auf einen Puffer, der die neue Umgebung enthält.

Count: Bestimmt die Zahl der Bytes, die kopiert werden sollen.

Rückgabewert Bei erfolgreicher Ausführung wird die Anzahl der Bytes zurückgegeben, die tatsächlich in die Umgebung kopiert wurden. Der Rückgabewert ist 0, wenn ein Fehler auftritt, und -1, wenn die Umgebung gelöscht ist.

Hinweis Das erste Feld im Puffer *Environ* muß dasselbe sein wie das, das mit dem Parameter *DeviceName* der Funktion *CreateDC* übergeben wurde. Bezeichnet *PortName* einen Nullanschluß (wie in der Datei WIN.INI vereinbart), wird die Gerätebezeichnung, auf die *Environ* zeigt, verwendet, um die gewünschte Umgebung anzulegen.

SetErrorMode

Syntax
```
function SetErrorMode(Mode: Word): Bool;
```

Beschreibung Diese Funktion steuert, ob Windows Fehler des DOS-Interrupt 24H übernimmt oder ob dies der aufrufenden Anwendung überlassen wird. Windows fängt alle Fehler des INT 24H ab. Dabei gibt es zwei Möglichkeiten:

- Wenn die Anwendung *SetErrorMode* mit auf 0 gesetztem Parameter *Mode* aufruft und anschließend ein Fehler des INT 24H auftritt, stellt Windows ein Fehlermeldungsfenster dar.

- Wenn die Anwendung *SetErrorMode* mit auf 1 gesetztem Parameter *Mode* aufruft und ein Fehler des INT 24H auftritt, stellt Windows das Standard-Fehlermeldungsfenster für den DOS-Interrupt 24H nicht dar, sondern gibt den originalen Fehlerrückruf des INT 21H an die Anwendung weiter. Dies ermöglicht es der Anwendung, Diskettenfehler unter Verwendung von INT 21H, AH=59H (Erweiterten Fehlercode holen) geeignet zu bearbeiten.

Mode: Bestimmt das Fehlermodus-Flag. Ist *Mode* auf 0 gesetzt, stellt Windows ein Fehlermeldungsfenster dar, wenn ein Fehler INT 24H auftritt. Ist *Mode* auf 1 gesetzt, gibt Windows den Fehlerrückruf des INT 21H an die aufrufende Anwendung weiter und stellt kein Meldungsfenster dar.

Rückgabewert Der Rückgabewert ist *True*, wenn *Mode* auf 1 gesetzt ist, andernfalls *False*.

SetFocus

Syntax
```
function SetFocus(Wnd: HWnd): HWnd;
```

Beschreibung Diese Funktion weist dem vom Parameter *Wnd* bezeichneten Fenster den Fokus zu. Der Fokus leitet alle nachfolgenden Tastatureingaben an das angegebene Fenster. Das Fenster, das vorher den Fokus hatte, verliert ihn. Ist *Wnd* 0, werden Tasteneingaben ignoriert.

SetFocus sendet eine Botschaft *wm_KillFocus* an das Fenster, das den Fokus verliert und eine Botschaft *wm_SetFocus* an das Fenster, das den Fokus erhält. Sie aktiviert auch entweder das Fenster, das den Fokus empfängt, oder dessen übergeordnetes Fenster.

Wnd: Bezeichnet das Fenster, das den Fokus erhalten soll.

Rückgabewert Der Rückgabewert bezeichnet das Fenster, das vorher den Fokus hatte. Er ist 0, wenn es kein solches Fenster gibt.

Hinweis Wenn ein Fenster aktiv ist, aber den Fokus nicht hat (das bedeutet, daß kein Fenster den Fokus hat), produziert jeder Tastendruck eine Botschaft *wm_SysChar, wm_SysKeyDown* oder *wm_SysKeyUp*.

Ist die Taste *vk_Menu* ebenfalls gedrückt, ist das Bit 30 im Parameter *lParam* der Botschaft gesetzt, andernfalls ist dieses Bit in den erzeugten Botschaften nicht gesetzt.

SetHandleCount

Syntax
```
function SetHandleCount(Number: Word): Word;
```

Beschreibung Diese Funktion ändert die Anzahl der für eine Task verfügbaren Datei-Handle. Standardmäßig ist die Zahl der für eine Task verfügbaren Datei-Handle 20.

Number: Bestimmt die Zahl der Datei-Handle, die von der Task verwendet werden können. Das Maximum ist 255.

Rückgabewert Es wird die Zahl der tatsächlich verfügbaren Datei-Handle für die Task zurückgeliefert. Diese kann kleiner sein als die Zahl, die vom Parameter *Number* vorgegeben ist.

SetKeyboardState

Syntax
```
procedure SetKeyboardState(var KeyState:
TKeyboardState);
```

Beschreibung Diese Prozedur kopiert die 256 Bytes im Parameter *KeyState* in die Tastaturstatus-Tabelle von Windows.

KeyState: Ein Array von 256 Bytes, das den Status der einzelnen Tasten enthält.

Hinweis Eine Anwendung sollte zuerst die Prozedur *GetKeyboardState* aufrufen, um das 256-Byte-Array zu initialisieren, und erst danach die gewünschten Bytes ändern.

Die Funktion *SetKeyboardState* setzt die LEDs und die BIOS-Flags für die Tasten NumLock , CapsLock und ScrollLock entsprechend des Arretierungszustandes der Einträge *vk_NumLock, vk_Capital* und *vk_OEM_Scroll* im Array.

Querverweis *GetKeyboardState*

SetMapMode

Syntax
```
function SetMapMode(DC: HDC; MapMode: Integer): Integer;
```

Beschreibung Diese Funktion stellt den Abbildungsmodus des angegebenen Gerätekontextes ein.

Der Abbildungsmodus definiert die Maßeinheit, die bei der Übersetzung virtueller Einheiten in gerätebezogene Einheiten benutzt wird und bestimmt die Ausrichtung der x- und y-Achsen des Geräts.

Die GDI benutzt den Abbildungsmodus, um virtuelle Koordinaten in passende gerätebezogene Koordinaten umzuwandeln.

DC: Bezeichnet den Gerätekontext.

MapMode: Bestimmt den neuen Abbildungsmodus. Er kann einer der in Kapitel 1 »Windows-Stile und -Konstanten« im Abschnitt »mm_Xxxx« beschriebenen Werte sein.

Rückgabewert Es wird der vorherigen Abbildungsmodus zurückgeliefert.

Hinweis Der Modus *mm_Text* erlaubt es Anwendungen, mit geräteeigenen Pixeln zu arbeiten, deren Größe von Gerät zu Gerät variiert.

Die Modi *mm_HiEnglish, mm_HiMetric, mm_LoEnglish, mm_LoMetric* und *mm_Twips* sind nützlich für Anwendungen, die in physikalisch geltenden Einheiten zeichnen müssen (wie zum Beispiel in Zoll oder Millimetern).

Der Modus *mm_Isotropic* stellt ein Seitenverhältnis 1:1 her. Er wird dann verwendet, wenn die exakte Form eines Bildes erhalten werden soll.

Der Modus *mm_Anisotropic* erlaubt es, die x- und y-Koordinaten unabhängig voneinander einzustellen.

SetMapperFlags

Syntax
```
function SetMapperFlags(DC: HDC; Flag: LongInt):
LongInt;
```

Beschreibung Diese Funktion wechselt den Algorithmus, den der Schriftenabbilder verwendet, um virtuelle Schriften auf physikalische Schriften abzubilden.

Ist das erste Bit des Parameters *Flag* auf 1 gesetzt, wählt der Abbilder nur Schriften, deren x-und y-Abmessungen genau mit denen des angegebenen Geräts übereinstimmen. Sind keine Schriften mit übereinstimmender Höhen- und Seitenabmessung vorhanden, wählt die GDI ein Höhen- und Seitenverhältnis, das mit dem von ihr gewählten übereinstimmt.

DC: Bezeichnet den Gerätekontext, der das Schriftenabbilder-Flag enthält.

Flag: Bestimmt, ob der Schriftenabbilder versucht, das Höhen- und Seitenverhältnis an das Gerät anzugleichen. Wenn das erste Bit auf 1 gesetzt ist, wählt der Abbilder nur Schriften aus, deren x- und y-Abmessungen exakt mit denen des angegebenen Geräts übereinstimmen.

Rückgabewert Es wird der vorherige Wert des Schriftenabbilder-Flags zurückgeliefert.

Hinweis Die übrigen Bits des Parameters *Flag* müssen 0 sein.

SetMenu

Syntax
```
function SetMenu(Wnd: HWnd; Menu: HMenu): Bool;
```

Beschreibung Diese Funktion ändert das Menü des angegebenen Fensters in das vom Parameter *Menu* bezeichnete Menü. Ist *Menu* 0, wird das aktuelle Menü des Fensters vom Bildschirm gelöscht. *SetMenu* bewirkt, daß das Fenster neu gezeichnet wird, um die Menüänderung widerzuspiegeln.

Wnd: Bezeichnet das Fenster, dessen Menü geändert werden soll.

Menu: Bezeichnet das neue Menü.

Rückgabewert Der Rückgabewert ist *True*, wenn das Menü geändert wurde, andernfalls ist er *False*.

Hinweis *SetMenu* entfernt ein vorheriges Menü nicht. Um dies zu bewirken, sollte eine Anwendung die Funktion *DestroyMenu* aufrufen.

Querverweis *DestroyMenu*

SetMenuItemBitmaps

Syntax
```
function SetMenuItemBitmaps(Menu: HMenu; Position,
Flags: Word; BitmapUnchecked, BitmapChecked: HBitmap):
Bool;
```

Beschreibung Diese Funktion ordnet die angegebenen Bitmaps einer Menüoption zu. Je nachdem, ob die Menüoption markiert ist oder nicht, stellt Windows das entsprechende Bitmap neben der Menüoption dar.

Menu: Bezeichnet das zu bearbeitende Menü.

Position: Bestimmt die Menüoption, die verändert werden soll. Wenn *Flags* auf *mf_ByPosition* gesetzt ist, bestimmt *Position* die Position der Menüoption, die erste Option im Menü steht an Position 0. Wenn *Flags* auf *mf_ByCommand* gesetzt ist, bestimmt *Position* die Befehls-ID der Menüoption.

Flags: Bestimmt die Interpretation des Parameters *Position*. Er kann auf *mf_ByCommand* (Voreinstellung) oder *mf_ByPosition* gesetzt werden.

BitmapUnchecked: Bezeichnet das Bitmap, das dargestellt werden soll, wenn die Menüoption nicht markiert ist.

BitmapChecked: Bezeichnet das Bitmap, das dargestellt werden soll, wenn die Menüoption markiert ist.

Rückgabewert Der Rückgabewert ist *True*, wenn die Funktion erfolgreich war, andernfalls ist er *False*.

Hinweis Ist einer der Parameter *BitmapUnchecked* oder *BitmapChecked* 0, stellt Windows für das zugehörige Attribut nichts neben der Menüoption dar. Sind beide Parameter 0, verwendet Windows die voreingestellte Auswahlmarkierung, wenn die Option markiert wird und löscht die Auswahlmarkierung, wenn die Option nicht markiert ist. Wenn das Menü entfernt und freigegeben wird, werden diese Bitmaps nicht freigegeben, hierfür ist die Anwendung verantwortlich.

Von der Funktion *GetMenuCheckMarkDimensions* werden die Dimensionen der für Menüoptionen benutzten Standardauswahlmarkierung abgerufen. Die Anwendung sollte diese Werte benutzen, um die geeignete Größe für Bitmaps, die mit dieser Funktion gesetzt werden, zu bestimmen.

SetMessageQueue

Syntax
```
function SetMessageQueue(Msg: Integer): Bool;
```

Beschreibung Diese Funktion erzeugt eine neue Botschafts-Warteschlange. Sie ist nützlich in Anwendungen, die eine Warteschlange für

mehr als acht Botschaften benötigen (Maximalgröße der Standard-Warteschlange). Der Parameter *Msg* bestimmt die Größe der neuen Warteschlange. Die Funktion muß von der Funktion *WinMain* einer Anwendung aufgerufen werden, bevor Fenster erzeugt und Botschaften gesendet werden. Die Funktion *SetMessageQueue* entfernt die alte Warteschlange zusammen mit allen Botschaften, die diese eventuell enthält.

Msg: Bestimmt die maximale Anzahl von Botschaften, die die neue Warteschlange enthalten kann.

Rückgabewert Der Rückgabewert ist *True*, wenn die Funktion eine neue Warteschlange erstellt hat, andernfalls ist er *False*.

Hinweis Ist der Rückgabewert 0, hat die Anwendung keine Warteschlange mehr, da die Funktion *SetMessageQueue* die ursprüngliche Warteschlange löscht, bevor sie versucht, eine neue zu erzeugen. Die Anwendung muß *SetMessageQueue* solange mit einer kleineren Warteschlangen-Größe aufrufen, bis die Funktion einen Wert ungleich 0 zurückgibt.

SetMetaFileBits

Syntax
```
function SetMetaFileBits(Mem: THandle): THandle;
```

Beschreibung Diese Funktion erstellt ein Speicher-Metafile aus den Daten in dem durch den Parameter *Mem* bezeichneten Speicherblock.

Mem: Bezeichnet den globalen Speicherblock, der die Metafile-Daten enthält. Es wird angenommen, daß die Daten mit der Funktion *GetMetaFileBits* erzeugt wurden.

Rückgabewert Bei erfolgreicher Ausführung der Funktion wird ein Speicher-Metafile zurückgeliefert, andernfalls ist der Rückgabewert 0.

Hinweis Nachdem die Funktion *SetMetaFileBits* zurückgekehrt ist, sollte für Bezüge auf das Metafile das zurückgelieferte anstelle des durch den Parameter *Mem* bezeichneten Handles verwendet werden.

SetPaletteEntries

Syntax
```
function SetPaletteEntries(Palette: HPalette;
StartIndex, NumEntries: Word; var PaletteEntries): Word;
```

Beschreibung Diese Funktion setzt RGB-Farbwerte und Flags in einer Reihe von Einträgen in einer virtuellen Palette.

Palette: Bezeichnet die virtuelle Palette.

StartIndex: Bestimmt den ersten Eintrag in der virtuellen Palette, der gesetzt werden soll.

NumEntries: Bestimmt die Anzahl der Einträge in der virtuellen Palette, die gesetzt werden sollen.

PaletteEntries: Ein Array von *TPaletteEntry*-Datenstrukturen, das RGB-Werte und Flags enthält.

Rückgabewert Bei erfolgreicher Ausführung wird die Zahl der Einträge, die in der virtuellen Palette gesetzt wurden, zurückgeliefert, andernfalls 0.

Hinweis Wenn die virtuelle Palette in einen Gerätekontext ausgewählt wurde, während die Anwendung *SetPaletteEntries* aufruft, haben die Veränderungen solange keine Auswirkung, bis die Anwendung *RealizePalette* aufruft.

SetParent

Syntax
```
function SetParent(WndChild, WndNewParent: HWnd): HWnd;
```

Beschreibung Diese Funktion ändert das übergeordnete Fenster eines untergeordneten Fensters. Ist das vom Parameter *WndChild* bezeichnete Fenster sichtbar, führt Windows eine entsprechende Neudarstellung durch.

WndChild: Bezeichnet das untergeordnete Fenster.

WndNewParent: Bezeichnet das übergeordnete Fenster.

Rückgabewert Es wird das vorherige übergeordnete Fenster zurückgeliefert.

SetPixel

Syntax
```
function SetPixel(DC: HDC; X, Y: Integer; Color:
TColorRef): LongInt;
```

Beschreibung Diese Funktion setzt ein Pixel an dem durch die Parameter *X* und *Y* bestimmten Punkt mit der nächstmöglichen Annäherung an den im Parameter *Color* vorgegebenen Farbwert.

Der Punkt muß innerhalb der Clipping-Region liegen, ist dies nicht der Fall, wird die Funktion ignoriert.

DC: Bezeichnet den Gerätekontext.

X: Bestimmt die virtuelle x-Koordinate des zu setzenden Punktes.

Y: Bestimmt die virtuelle y-Koordinate des zu setzenden Punktes.

Color: TColorRef-Struktur mit der Darstellungsfarbe für den Punkt.

Rückgabewert Der Rückgabewert gibt den RGB-Farbwert für die Farbe an, in der der Punkt tatsächlich gezeichnet wurde. Der Wert kann

sich von dem durch den Parameter *Color* vorgegebenen unterscheiden, wenn ein Annäherungswert für dieser Farbe benutzt wird. Wenn die Funktion fehlschlägt (wenn der Punkt außerhalb der Clipping-Region liegt), ist der Rückgabewert -1.

Hinweis Nicht alle Geräte unterstützen diese Funktion (siehe die Beschreibung der Funktion *GetDeviceCaps*).

SetPolyFillMode

Syntax
```
function SetPolyFillMode(DC: HDC; PolyFillMode:
Integer): Integer;
```

Beschreibung Diese Funktion setzt den Polygonfüllmodus für GDI-Funktionen, die den Polygon-Algorithmus einsetzen, um innenliegende Punkte zu berechnen.

DC: Bezeichnet den Gerätekontext.

PolyFillMode: Bestimmt den neuen Füllmodus. Der Parameter kann die Werte *Alternate* und *Winding* annehmen. Einzelheiten hierzu finden Sie im folgenden Hinweis.

Rückgabewert Bei erfolgreicher Ausführung wird der vorherigen Füllmodus zurückgegeben, andernfalls ist der Rückgabewert 0.

Hinweis Im allgemeinen unterscheiden sich die Modi nur in den Fällen, in denen ein komplexes, überlappendes Polygon ausgefüllt werden muß (zum Beispiel ein fünfseitiges Polygon, das einen fünfzackigen Stern mit einem Fünfeck in der Mitte bildet). In solchen Fällen füllt der Modus *Alternate* alternierend die einzelnen abgeschlossenen Bereiche innerhalb des Polygons (also die Zacken des Sterns), der Modus *Winding* dagegen füllt den gesamten Bereich aus (hier: die Sternzacken und das Fünfeck).

Wenn der Füllmodus *Alternate* ist, füllt die GDI den Bereich zwischen geradzahligen und ungeradzahligen Polygon-Seiten auf jeder Scan-Zeile. Das heißt, die GDI füllt den Bereich zwischen der ersten und der zweiten Seite, zwischen der dritten und vierten Seite, und so weiter.

Zum Ausfüllen aller Bereiche bewirkt der Modus *Winding*, daß die GDI eine Umrandung mit nicht überlappenden Seiten berechnet und zeichnet. Zum Beispiel wird das fünfseitige Polygon, das den Stern bildet, als zehnseitiges Polygon mit nicht überlappenden Seiten gezeichnet; der entstehende Stern wird ausgefüllt.

SetProp

Syntax
```
function SetProp(Wnd: HWnd; Str: PChar; Data: THandle):
Bool;
```

Beschreibung Diese Funktion fügt einen neuen Eintrag zur Eigenschaftsliste des angegebenen Fensters hinzu oder verändert einen darin vorhandenen Eintrag.

Wenn der vom Parameter *Str* bezeichnete String noch nicht in der Liste existiert, fügt *SetProp* einen neuen Eintrag zu der Liste hinzu, der neue Eintrag enthält den String und das Handle. Andernfalls ersetzt die Funktion das aktuelle Handle des Strings durch *Data*.

Der Parameter *Data* kann jeden für die Anwendung nützlichen 16-Bit-Wert enthalten.

Wnd: Bezeichnet das Fenster, dessen Eigenschaftsliste den neuen Eintrag aufnehmen soll.

Str: Zeigt auf einen nullterminierten String oder ein Atom, das einen String bezeichnet. Wenn ein Atom gegeben ist, muß es vorher mit der Funktion *AddAtom* erstellt worden sein. Das Atom, ein 16-Bit-Wert, muß im niederwertigen Word von *Str* plaziert werden, das höherwertige Word muß 0 sein.

Data: Bezeichnet ein Daten-Handle, dessen zugehörige Daten in die Eigenschaftsliste kopiert werden soll.

Rückgabewert Der Rückgabewert ist *True*, wenn das Daten-Handle und der String zur Eigenschaftsliste hinzugefügt wurden, andernfalls ist er *False*.

Hinweis Bevor sie das Fenster entfernt und freigibt (also bevor sie die Botschaft *wm_Destroy* verarbeitet), ist die Anwendung für das Löschen aller Einträge verantwortlich, die sie zur Eigenschaftsliste hinzugefügt hat.

Die Funktion *RemoveProp* dient zum Löschen von Einträgen aus einer Eigenschaftsliste.

SetRect

Syntax
```
prozedure SetRect(var Rect: TRect; X1, Y1, X2, Y2:
Integer);
```

Beschreibung Diese Prozedur erzeugt ein neues Rechteck, indem sie die *TRect*-Datenstruktur im Parameter *Rect* ausfüllt. Die Koordinaten werden durch die Parameter *X1*, *Y1*, *X2* und *Y2* bestimmt.

Rect: Eine *TRect*-Datenstruktur, die die neuen Rechteckkoordinaten aufnehmen soll.

X1: Bestimmt die x-Koordinate der oberen linken Ecke.

Y1: Bestimmt die y-Koordinate der oberen linken Ecke.

X2: Bestimmt die x-Koordinate der unteren rechten Ecke.

Y2: Bestimmt die y-Koordinate der unteren rechten Ecke.

Hinweis Die Breite des Rechtecks, die durch den Absolutbetrag von *X2 - X1* gegeben ist, darf 32767 Einheiten nicht überschreiten. Dieser Grenzwert gilt auch für die Höhe des Rechtecks (*Y2 - Y1*).

SetRectEmpty

Syntax
```
procedure SetRectEmpty(var Rect: TRect);
```

Beschreibung Diese Prozedur erzeugt ein leeres Rechteck (alle Koordinaten sind 0).

Rect: Eine *TRect*-Datenstruktur, die das leere Rechteck aufnehmen soll.

SetRectRgn

Syntax
```
procedure SetRectRgn(Rgn: HRgn; X1, Y1, X2, Y2:
Integer);
```

Beschreibung Diese Prozedur erzeugt eine rechteckige Region. Im Gegensatz zu *CreateRectRgn* ruft sie jedoch nicht die lokale Speicherverwaltung auf, sondern benutzt stattdessen den Speicherplatz, der für die mit dem Parameter *Rgn* bezeichnete Region reserviert ist. Die mit den Parametern *X1*, *Y1, X2* und *Y2* bezeichneten Punkte bestimmen den minimalen Umfang des reservierten Speichers.

Rgn: Bezeichnet die Region.

X1: Bestimmt die x-Koordinate der oberen linken Ecke der rechteckigen Region.

Y1: Bestimmt die y-Koordinate der oberen linken Ecke der rechteckigen Region.

X2: Bestimmt die x-Koordinate der unteren rechten Ecke der rechteckigen Region.

Y2: Bestimmt die y-Koordinate der unteren rechten Ecke der rechteckigen Region.

Hinweis Diese Prozedur sollte anstelle der Funktion *CreateRectRgn* benutzt werden, um Aufrufe der lokalen Speicherverwaltung zu vermeiden.

Querverweis *CreateRectRgn*

SetResourceHandler

Syntax
```
function SetResourceHandler(Instance: THandle; ResType:
Pointer; LoadFunc: TFarProc): TFarProc;
```

Beschreibung Diese Funktion bereitet eine Funktion zum Laden von Ressourcen vor. Sie wird intern von Windows benutzt, um berechnete Ressourcen zu implementieren und kann nützlich sein für Anwendungen, die ihre eigenen Typen von Ressourcen bearbeiten, ihre Verwendung ist aber nicht unbedingt erforderlich.

Der Parameter *LoadFunc* zeigt auf eine von der Anwendung bereitgestellte Callback-Funktion. Diese erhält Informationen über die Ressource, die gesperrt werden soll, und kann sie wie gewünscht weiterverarbeiten. Nachdem sie zurückgekehrt ist, versucht *LockResource*, die Ressource noch einmal zu sperren.

Instance: Bezeichnet die Instanz des Moduls, dessen ausführbare Datei die Ressource enthält.

ResType: Zeigt auf einen Integer, der einen Ressourcentyp bezeichnet.

LoadFunc: Ist die Adresse der Prozedurinstanz der von der Anwendung bereitgestellten Callback-Funktion.

Rückgabewert Der Rückgabewert zeigt auf die von der Anwendung bereitgestellte Funktion.

Callback-Funktion

```
function LoadFunc(hMem, hInstance, HResInfo: THandle):
TFarProc; export;
```

LoadFunc: Platzhalter für die von der Anwendung gestellte Funktionsbezeichnung.

hMem: Bezeichnet eine gespeicherte Ressource.

hInstance: Bezeichnet die Instanz des Moduls, dessen ausführbare Datei die Ressource enthält.

hResInfo: Bezeichnet die Ressource. Es wird angenommen, daß die Ressource vorher mit der Funktion *FindResource* erzeugt wurde.

Hinweis Der Parameter *hMem* ist 0, wenn die Ressource noch nicht geladen war. Wenn ein Versuch, einen von *hMem* bezeichneten Block zu sperren, nicht gelingt, bedeutet das, daß die Ressource verworfen wurde und neu geladen werden muß.

Die Adresse der Dialogfunktion, die als Parameter *LoadFunc* übergeben wurde, muß mit der Funktion *MakeProcInstance* erzeugt werden.

SetROP2

Syntax

```
function SetROP2(DC: HDC; DrawMode: Integer): Integer;
```

Beschreibung Diese Funktion setzt den aktuellen Zeichenmodus. Die GDI benutzt den Zeichenmodus, um Stifte und die Innenflächen

von ausgefüllten Objekten mit den Farben zu kombinieren, die sich schon auf der Bildschirmoberfläche befinden. Der Modus legt fest, wie die Farbe des Stiftes oder des Flächeninneren und die Farbe, die sich schon auf der Bildschirmfläche befindet, zu einer neuen Farbe kombiniert werden.

DC: Bezeichnet den Gerätekontext.

DrawMode: Bestimmt den neuen Zeichenmodus. Er kann jeder der im Abschnitt »r2_Xxxx« in Kapitel 1 »Windows-Stile und -Konstanten« beschriebenen Werte annehmen.

Rückgabewert Es wird der vorherhige Zeichenmodus zurückgeliefert.

Hinweis Zeichenmodi legen fest, wie die GDI Quell- und Zielfarben kombiniert, wenn mit dem aktuellen Stift gezeichnet wird. Die Zeichenmodi sind eigentlich binäre Rasteroperationscodes, die alle möglichen booleschen Funktionen zweier Variablen repräsentieren. Dabei werden die binären Operationen **and**, **or**, **xor** (Exklusiv-Oder) und die einstellige Operation **not** verwendet. Der Zeichenmodus ist nur für Geräte mit Rasterpunkten geeignet, auf vektororientierten Geräten ist er nicht verfügbar. Weitere Informationen bietet die Beschreibung der Funktion *GetDeviceCaps* sowie der Abschnitt »r2_Xxxx« in Kapitel 1 »Windows-Stile und -Konstanten«.

SetScrollPos

Syntax
```
function SetScrollPos(Wnd: HWnd; Bar, Pos: Integer;
Redraw: Bool): Integer;
```

Beschreibung Diese Funktion setzt die aktuelle Position des Bildlaufleistensymbols auf die durch den Parameter *Pos* angegebene Position und zeichnet die Bildlaufleiste neu (wenn dies angegeben ist), um die neue Position wiederzugeben.

Wnd: Bezeichnet das Fenster, dessen Bildlaufleiste gesetzt werden soll.

Bar: Bestimmt die zu setzende Bildlaufleiste. Er kann einen der folgenden Werte annehmen:

Wert	Bedeutung
sb_Ctl	Setzt die Position einer selbständigen Bildlaufleiste. In diesem Fall muß der Parameter *Wnd* das Handle einer Bildlaufleiste sein.
sb_Horz	Setzt die Position der horizontalen Bildlaufleiste eines Fensters.
sb_Vert	Setzt die Position der vertikalen Bildlaufleiste eines Fensters.

Pos: Bestimmt die neue Position. Sie muß innerhalb des Bildlaufbereichs liegen.

Redraw: Bestimmt, ob die Bildlaufleiste neu gezeichnet werden soll, um die neue Position widerzuspiegeln. Ist der Parameter *Redraw True*, wird die Bildlaufleiste neu gezeichnet, ist er *False*, so ist dies nicht der Fall.

Rückgabewert Es wird die vorherige Position des Bildlaufleistensymbols zurückgeliefert.

Hinweis Wenn die Bildlaufleiste von einem nachfolgenden Aufruf einer anderen Funktion neu gezeichnet wird, sollte der Parameter *Redraw* auf *False* gesetzt werden.

SetScrollRange

Syntax
```
procedure SetScrollRange(Wnd: HWnd; Bar, MinPos,
MaxPos: Integer; Redraw: Bool);
```

Beschreibung Diese Prozedur setzt Minimal- und Maximalpositionswerte für die gegebene Bildlaufleiste. Sie kann außerdem dazu eingesetzt werden, Standardbildlaufleisten anzuzeigen oder zu verbergen. Hierzu müssen die Parameter *MinPos* und *MaxPos* auf 0 gesetzt werden.

Wnd: Bezeichnet ein Fenster oder eine Bildlaufleiste in Abhängigkeit vom Wert des Parameters *Bar*.

Bar: Bestimmt die zu setzende Bildlaufleiste. Er kann einen der folgenden Werte annehmen:

Wert	Bedeutung
sb_Ctl	Setzt den Bereich einer selbständigen Bildlaufleiste. In diesem Fall muß der Parameter *Wnd* das Handle auf eine Bildlaufleiste sein.
sb_Horz	Setzt den Bereich der horizontalen Bildlaufleiste eines Fensters.
sb_Vert	Setzt den Bereich der vertikalen Bildlaufleiste eines Fensters.

MinPos: Legt die kleinste Position im Bildlauf fest.

MaxPos: Legt die größte Position im Bildlauf fest.

Redraw: Legt fest, ob die Bildlaufleiste neu gezeichnet werden soll, um die Änderung anzuzeigen. Ist der Parameter *Redraw True*, wird die Bildlaufleiste neu gezeichnet, andernfalls nicht.

Hinweis Eine Anwendung sollte diese Funktion nicht zum Ausblenden einer Bildlaufleiste aufrufen, während sie eine Bildlaufleistenbotschaft verarbeitet. Wenn *SetScrollRange* unmittelbar auf die

Funktion *SetScrollPos* folgt, sollte der Parameter *Redraw* in *SetScrollPos* auf *False* gesetzt werden, um zu verhindern, daß die Bildlaufleiste zweimal gezeichnet wird.

Die Differenz zwischen den durch die Parameter *MinPos* und *MaxPos* festgelegten Werten darf nicht größer als 32767 sein.

SetSoundNoise

Syntax

```
function SetSoundNoise(Source, Duration: Integer):
Integer;
```

Beschreibung

Diese Funktion setzt Quelle und Dauer eines Tons am Lautsprecher des Klangausgabegeräts.

Source: Legt die Tonquelle fest. Er kann einen der folgenden Werte annehmen, wobei *N* der Wert für eine gewünschte Frequenz ist:

Wert	Bedeutung
s_Period512	Quellfrequenz ist N/512 (hoher Ton); weniger heller Ton.
s_Period1024	Quellfrequenz ist N/1024.
s_Period2048	Quellfrequenz ist N/2048 (tiefer Ton); hellerer Ton.
s_PeriodVoice	Quellfrequenz von Voice-Kanal 3.
s_White512	Quellfrequenz ist N/512 (hoher Ton); weniger heller Ton.
s_White1024	Quellfrequenz ist N/1024.
s_White2048	Quellfrequenz ist N/2048 (tiefer Ton); hellerer Ton.
s_WhiteVoice	Quellfrequenz von Voice-Kanal 3.

Duration: Legt die Dauer des Tons fest (in Clock-Ticks).

Rückgabewert

Der Rückgabewert ist 0, wenn die Funktion erfolgreich war. Ist die Quelle ungültig, ist der Rückgabewert *s_SerDSR*.

SetStretchBltMode

Syntax

```
function SetStretchBltMode(DC: HDC; StretchMode:
Integer): Integer;
```

Beschreibung

Diese Funktion setzt den Streckmodus für die *StretchBlt*-Funktion. Der Streckmodus legt fest, welche Scan-Zeilen und/oder -Spalten *StretchBlt* beim Kontrahieren eines Bitmaps entfernt.

DC: Bezeichnet den Gerätekontext.

StretchMode: Bestimmt den neuen Streckmodus. Er kann einen der folgenden Werte annehmen:

Wert	Bedeutung
BlackOnWhite	**and** in den getilgten Zeilen. Dieser Modus erhält schwarze Pixel auf Kosten der weißen durch den Gebrauch des Operators **and** mit den zu tilgenden und den verbleibenden Zeilen.
ColorOnColor	Dieser Modus löscht alle getilgten Zeilen und versucht dabei nicht, ihre Information zu erhalten.
WhiteOnBlack	**or** in den getilgten Zeilen. Dieser Modus erhält weiße Pixel auf Kosten der schwarzen durch den Gebrauch des Operators **or** mit den zu tilgenden und den verbleibenden Zeilen.

Ein typischer Verwendungszweck der Modi *BlackOnWhite* und *WhiteOnBlack* ist es, Vordergrundpixel in Monochrom-Bitmaps zu erhalten. Eine typische Verwendung des Modus *ColorOnColor* ist es, die Farben in Farb-Bitmaps zu erhalten.

Rückgabewert Der Rückgabewert gibt den vorherigen Streckmodus an. Er kann *BlackOnWhite, ColorOnColor* oder *WhiteOnBlack* sein.

SetSwapAreaSize

Syntax
```
function SetSwapAreaSize(Size: Word): LongInt;
```

Beschreibung Diese Funktion erhöht den Speicherumfang, den eine Anwendung für ihre Code-Segmente verwendet. Der maximal verfügbare Speicherumfang ist die Hälfte des freien Speichers, der nach dem Laden von Windows verbleibt.

Size: Bestimmt die Anzahl der 16-Byte-Paragraphen, die eine Anwendung zum Gebrauch als Code-Segment anfordert.

Rückgabewert Das niederwertige Word des Rückgabewertes gibt die Anzahl der erhaltenen Paragraphen an, die als Speicher für das Code-Segment belegt werden (oder die gegenwärtige Größe, wenn *Size* 0 ist); das höherwertige Word gibt die maximal verfügbare Größe an.

Hinweis Legt *Size* eine Größe fest, die die verfügbare überschreitet, setzt diese Funktion die Größe auf den maximal verfügbaren Wert.

Sobald Speicher für den Gebrauch als Code-Segment zugeordnet wurde, kann ihn eine Anwendung nicht mehr durch den Aufruf der Funktion *GlobalAlloc* als Datensegment nutzen.

Der Aufruf von *SetSwapAreaSize* verbessert die Leistungsfähigkeit einer Anwendung, indem sie verhindert, daß Speicherbereiche überschrieben werden. Er vermindert jedoch den verfügbaren Speicher für Datenobjekte und kann die Leistungsfähigkeit anderer Anwendungen einschränken. Vor

dem Aufruf von *SetSwapAreaSize* sollte eine Anwendung *GetNumTasks* aufrufen, um festzustellen, wieviele andere Tasks gerade laufen.

SetSysColors

Syntax

```
procedure SetSysColors(Changes: Integer; var SysColor:
Integer; var ColorValues);
```

Beschreibung

Diese Prozedur setzt die Systemfarben für ein oder mehrere Bildschirmelemente. Bildschirmelemente sind die verschiedenen Teile eines Fensters und der Windows-Anzeige, die am Bildschirm des Systems erscheinen. Die Prozedur *SetSysColors* ändert die Anzahl der durch den Parameter *Changes* festgelegten Elemente. Sie verwendet dazu den Farben- und Systemfarbenindex, der in den Arrays der Parameter *SysColor* und *ColorValues* enthalten ist.

SetSysColors sendet eine *wm_SysColorChange*-Botschaft an alle Fenster, um sie über den Farbwechsel zu informieren. Außerdem weist sie Windows an, die betroffenen Teile aller momentan sichtbaren Fenster neu zu zeichnen.

Changes: Bestimmt die Anzahl der Systemfarben, die geändert werden sollen.

SysColor: Ein Array von Integerindizes, die die zu ändernden Elemente bestimmen. Die Indexwerte, die verwendet werden können, sind im folgenden Hinweis beschrieben.

ColorValues: Ein Array vorzeichenloser Integer-Zahlen, das die neuen RGB-Farbwerte für jedes Element enthält.

Hinweis

SetSysColors ändert nur die interne Systemliste. Sie ändert nicht den Abschnitt [COLORS] der Initialisierungsdatei von Windows, WIN.INI. Änderungen gelten nur für die momentane Windows-Sitzung. Systemfarben sind eine gemeinsam benutzte Ressource. Eine Anwendung sollte eine Farbe nicht ändern, wenn diese nicht in allen Fenstern aller momentan laufenden Anwendungen geändert werden soll. Systemfarben für Monochrombildschirme werden gewöhnlich als verschiedene Graustufen interpretiert. Im folgenden eine Beschreibung der Werte für *SysColor*.

Wert	Bedeutung
color_ActiveBorder	Aktive Fenstergrenze.
color_ActiveCaption	Aktive Überschrift.
color_AppWorkSpace	Hintergrundfarbe von Multidokumentschnittstellenanwendungen (MDI).
color_Background	Desktop.

Wert	Bedeutung
color_BtnFace	Flächenschattierung bei Aktionsschaltern.
color_BtnShadow	Rahmenschattierung bei Aktionsschaltern.
color_BtnText	Text in Aktionsschaltern.
color_CaptionText	Text in Titel, Bildgrößenfeld, Bildlaufpfeilfeld.
color_GrayText	Grauer (nichtaktivierbarer) Text. Diese Farbe wird auf 0 gesetzt, wenn der momentane Bildschirmtreiber einen echten Grauton nicht unterstützt.
color_Highlight	Kennzeichnet die gewählte Option in einem Steuerelement.
color_HighlightText	Text einer gewählten Option in einem Steuerelement.
color_InactiveBorder	Grenze eines deaktivierten Fensters.
color_InactiveCaption	Titel eines deaktivierten Fensters.
color_Menu	Menühintergrund.
color_MenuText	Text in Menüs.
color_ScrollBar	Grauer Bereich einer Bildlaufleiste.
color_Window	Fensterhintergrund.
color_WindowFrame	Fensterrahmen.
color_WindowText	Text in Fenstern.

SetSysModalWindow

Syntax
```
function SetSysModalWindow(Wnd: HWnd): HWnd;
```

Beschreibung Diese Funktion macht das gewählte Fenster zu einem systemmodalen Fenster.

Wnd: Bezeichnet das Fenster, das systemmodal gemacht werden soll.

Rückgabewert Der Rückgabewert gibt das Fenster an, das vorher das systemmodale Fenster war.

Hinweis Wenn ein anderes Fenster als das angegebene zum aktiven Fenster gemacht wird (das systemmodale Fenster öffnet beispielsweise ein Dialogfenster, das zum aktiven Fenster wird), so wird das aktive Fenster zum systemmodalen Fenster. Wird das Originalfenster wieder aktiv, ist es systemmodal, um den systemmodalen Status aufzuheben, muß das systemmodale Fenster geschlossen werden.

SetSystemPaletteUse

Syntax
```
function SetSystemPaletteUse(DC: HDC; Usage: Word):
Word;
```

Beschreibung Diese Funktion erlaubt einer Anwendung, die volle Systempalette zu verwenden. Die Systempalette enthält per Voreinstellung 20 statische Farben, die nicht geändert werden, wenn eine Anwendung ihre virtuelle Palette realisiert.

Der durch den Parameter *DC* festgelegte Gerätekontext muß auf ein Gerät verweisen, das Farbpaletten unterstützt.

DC: Bezeichnet den Gerätekontext.

Usage: Bestimmt den neuen Gebrauch der Systempalette. Er kann einen der folgenden Werte annehmen:

Wert	Bedeutung
syspal_NoStatic	Die Systempalette enthält keine statischen Farben außer Schwarz und Weiß.
syspal_Static	Die Systempalette enthält statische Farben, die sich nicht ändern, wenn eine Anwendung ihre virtuelle Palette realisiert.

Rückgabewert Der Rückgabewert gibt den vorhergehenden Gebrauch der Systempalette an, er ist entweder *syspal_NoStatic* oder *syspal_Static*.

Hinweis Eine Anwendung darf diese Funktion darf nur aufgerufen, wenn ihr Fenster den Fokus besitzt.

Ruft eine Anwendung *SetSystemPaletteUse* auf, wobei *Usage* auf *syspal_NoStatic* gesetzt ist, behält Windows weiterhin zwei Einträge in der Systempalette für reines Weiß bzw. reines Schwarz bei.

Wenn *Usage* auf *syspal_NoStatic* gesetzt ist, so müssen Sie nach dem Aufruf von *SetSystemPaletteUse* folgende Maßnahmen durchführen:

1. Rufen Sie *UnrealizeObject* auf, damit die GDI die virtuelle Palette komplett neu anlegt.

2. Realisieren Sie die virtuelle Palette.

3. Rufen Sie *GetSysColors* auf, um die gegenwärtigen Systemfarbeinstellungen zu sichern.

4. Rufen Sie *SetSysColors* auf, um die Systemfarben auf sinnvolle Werte zu setzen. Benachbarte oder überlappende Elemente (wie Fensterrahmen und -grenzen) sollten beispielsweise auf schwarz oder weiß gesetzt werden.

5. Geben Sie die Rundrufbotschaft *wm_SysColorChange* aus, um anderen Fenstern die Verwendung der neuen Systemfarben zu ermöglichen.

Wenn das Fenster der Anwendung den Fokus verliert oder schließt, muß die Anwendung folgende Schritte ausführen:

1. Rufen Sie *SetSystemPaletteUse* auf, wobei der Parameter *Usage* auf *syspal_Static* gesetzt wird.

2. Rufen Sie *UnrealizeObject* auf, damit die GDI die virtuelle Palette bei ihrem Aufruf komplett neu anlegt.

3. Realisieren Sie die virtuelle Palette.

4. Setzen Sie die Systemfarben auf ihre ursprünglichen Werte.

5. Geben Sie die Rundrufbotschaft *wm_SysColorChange* aus.

Querverweis *GetSysColor, SetSysColors, UnrealizeObject, wm_SysColorChange*

SetTextAlign

Syntax
```
function SetTextAlign(DC: HDC; Flags: Word): Word;
```

Beschreibung Diese Funktion setzt die Textausrichtungsflags für den gegebenen Gerätekontext. Die Funktionen *TextOut* und *ExtTextOut* verwenden diese Flags beim Positionieren eines Textstrings auf dem Bildschirm oder auf einem Gerät. Die Flags legen das Verhältnis zwischen einem speziellen Punkt und einem Rechteck, das den Text umschließt, fest. Die Koordinaten dieses Punktes werden als Parameter an die Funktion *TextOut* übergeben. Das Rechteck, das den Text umschließt, wird durch die aufeinanderfolgenden Zeichenzellen im Textstring gebildet.

DC: Bezeichnet den Gerätekontext.

Flags: Bestimmt eine Maske mit Werten der folgenden Liste. Für horizontale und vertikale Ausrichtung darf nur jeweils ein Flag gewählt werden. Außerdem kann nur eines der beiden Flags, die die aktuelle Position beeinflussen, gewählt werden:

Wert	Bedeutung
ta_Baseline	Bestimmt die Ausrichtung des Punktes an der Grundlinie der gewählten Schrift.
ta_Botton	Bestimmt die Ausrichtung des Punktes an der Unterseite des umschließenden Rechtecks.
ta_Center	Bestimmt die Ausrichtung des Punktes am horizontalen Mittelpunkt des umschließenden Rechtecks.
ta_Left	Bestimmt die Ausrichtung des Punktes an der linken Seite des umschließenden Rechtecks.
ta_NoUpdateCP	Bestimmt, daß die aktuelle Position nicht nach jedem Aufruf der Funktionen *TextOut* oder *ExtTextOut* aktualisiert wird.
ta_Right	Bestimmt die Ausrichtung des Punktes an der rechten Seite des umschließenden Rechtecks.

Wert	Bedeutung
ta_Top	Bestimmt die Ausrichtung des Punktes an der Oberseite des umschließenden Rechtecks.
ta_UpDateCP	Bestimmt, daß die aktuelle Position nach jedem Aufruf der Funktionen *TextOut* oder *ExtTextOut* aktualisiert wird.

Die Standardwerte sind *ta_Left, ta_Top,* und *ta_NoUpdateCP.*

Rückgabewert Der Rückgabewert gibt die vorherige Einstellung für die Textausrichtung an, das niederwertige Word enthält die horizontale, das höherwertige die vertikale Ausrichtung.

SetTextCharacterExtra

Syntax
```
function SetTextCharacterExtra(DC: HDC; CharExtra:
Integer): Integer;
```

Beschreibung Diese Funktion setzt den Wert für den Zeichenabstand. Die GDI fügt diesen Abstand an jedes Zeichen an – einschließlich der Unterbrechungszeichen – wenn im Gerätekontext eine Textzeile geschrieben wird.

DC: Bezeichnet den Gerätekontext.

CharExtra: Bestimmt die Größe des zusätzlichen Abstandes in virtuellen Einheiten, der an jedes Zeichen angefügt werden soll. Ist der aktuelle Abbildungsmodus nicht *mm_Text,* so wird der Parameter *CharExtra* übertragen und auf das nächste Pixel gerundet.

Rückgabewert Der Rückgabewert gibt den Wert des vorherigen Zeichenabstands an.

SetTextColor

Syntax
```
function SetTextColor(DC: HDC; Color: TColorRef):
LongInt;
```

Beschreibung Diese Funktion setzt die Textfarbe auf die Farbe, die durch den Parameter *Color* bestimmt ist oder auf die nächstliegende physikalische Farbe, wenn das Gerät die durch *Color* festgelegte Farbe nicht darstellen kann. Die GDI verwendet die Textfarbe, um das Erscheinungsbild jedes Zeichens zu bestimmen, das durch die Funktionen *TextOut* und *ExtTextOut* geschrieben wird. Die GDI verwendet die Textfarbe auch bei der Konvertierung von Bitmaps von Farbe in Monochrom und umgekehrt.

Die Hintergrundfarbe für ein Zeichen wird durch die Funktionen *SetBkColor* und *SetBkMode* bestimmt. Iin der Beschreibung der Funktionen *BitBlt* und *StretchBlt* finden Sie Informationen zur Konvertierung von Farb-Bitmaps.

DC: Bezeichnet den Gerätekontext.

Color: *TColorRef*-Struktur, die die Textfarbe bestimmt.

Rückgabewert Der Rückgabewert gibt den RGB-Farbwert der vorherigen Textfarbe an.

Querverweis *SetBkColor, SetBkMode*

SetTextJustification

Syntax
```
function SetTextJustification(DC: HDC; BreakExtra,
BreakCount: Integer): Integer;
```

Beschreibung Diese Funktion bereitet die GDI darauf vor, eine Textzeile zu justieren und dabei die durch die Parameter *BreakExtra* und *BreakCount* festgelegten Justierungsparameter zu verwenden. Um einen Text zu justieren, vergibt die GDI zusätzliche Pixel für die Unterbrechungszeichen in einer Textzeile, die von der Funktion *TextOut* geschrieben wird. Das Unterbrechungszeichen, das zur Abgrenzung von Worten verwendet wird, ist üblicherweise das Leerzeichen (ASCII 32), eine Schrift kann jedoch auch ein anderes Zeichen definieren. Mit der Funktion *GetTextMetrics* kann das Unterbrechungszeichen einer Schrift ermittelt werden.

SetTextJustification bereitet die Justierung vor, indem der Wert des anzufügenden Abstandes festgelegt wird. Der Parameter *BreakExtra* bestimmt den Gesamtbetrag des Abstandes (in virtuellen Einheiten), der an die Zeile angefügt wird. Der Parameter *BreakCount* bestimmt, wieviele Unterbrechungszeichen in der Zeile sind. Die nachfolgende Funktion *TextOut* verteilt den zusätzlichen Abstand gleichmäßig auf jedes Unterbrechungszeichen der Zeile.

Die Funktion *GetTextExtent* wird immer zusammen mit der Funktion *SetTextJustification* verwendet. *GetTextExtent* berechnet die Breite einer gegebenen Zeile vor der Justierung. Diese Breite muß bekannt sein, damit ein geeigneter Wert für *BreakExtra* berechnet werden kann.

SetTextJustification kann dazu verwendet werden, eine Zeile zu justieren, die mehrere Folgen in verschiedenen Schriften enthält. In diesem Fall muß die Zeile stückweise durch separate Justierung und Ausgabe jeder Folge erzeugt werden. Da während der Justierung Rundungsfehler auftreten können, verwaltet die GDI einen Rundungsfehlerwert, der den aktuellen Fehler bestimmt. Bei der Justierung einer Zeile, die mehrere

Folgen enthält, verwendet *GetTextExtent* automatisch diesen Fehlerwert beim Berechnen des Umfangs der nächsten Folge. Dadurch wird es *TextOut* ermöglicht, den Fehler für die neue Folge zu berücksichtigen. Nachdem eine Zeile vollständig justiert worden ist, muß dieser Fehlerausdruck gelöscht werden, um zu verhindern, daß er in die nächste Zeile übertragen wird. Dies geschieht dadurch, daß *SetTextJustification* mit auf 0 gesetztem *BreakExtra* aufgerufen wird.

DC: Bezeichnet den Gerätekontext.

BreakExtra: Bestimmt den Gesamtbetrag des Abstandes in virtuellen Einheiten, um den die Textzeile ergänzt wird. Ist der aktuelle Abbildungsmodus nicht *mm_Text,* wird der durch den Parameter *BreakExtra* festgelegte Wert übertragen und auf das nächste Pixel gerundet.

BreakCount: Bestimmt die Anzahl der Unterbrechungszeichen in der Zeile.

Rückgabewert Der Rückgabewert ist 1, wenn die Funktion erfolgreich war, ansonsten ist er 0.

Querverweis *GetTextExtent, GetTextMetrics, TextOut*

SetTimer

Syntax
```
function SetTimer(Wnd: HWnd; IDEvent: Integer; Elapse:
Word; TimerFunc: TFarProc): Word;
```

Beschreibung Diese Funktion erzeugt ein System-Timer-Ereignis. Tritt ein Timer-Ereignis ein, gibt Windows eine *wm_Timer*-Botschaft an die durch den Parameter *TimerFunc* bezeichnete anwendungsunterstützte Funktion aus, die dann das Ereignis bearbeiten kann. Hat *TimerFunc* dan Wert 0, werden *wm_Timer*-Botschaften in der Anwendungs-Warteschlange plaziert.

Wnd: Bestimmt das Fenster, das dem Timer zugeordnet werden soll. Ist *Wnd* 0, wird dem Timer kein Fenster zugeordnet.

IDEvent: Bestimmt einen von 0 verschiedenen Bezeichner für ein Timer-Ereignis, wenn der Parameter *Wnd* nicht 0 ist.

Elapse: Bestimmt die abgelaufene Zeit zwischen Timer-Ereignissen in Millisekunden.

TimerFunc: Ist die Adresse der Prozedurinstanz der Funktion, die benachrichtigt werden soll, wenn das Timer-Ereignis stattfindet. Wenn *TimerFunc* 0 ist, wird die *wm_Timer*-Botschaft in der Anwendungs-Warteschlange plaziert, und das *hwnd*-Element der *TMsg*-Struktur enthält den Parameter *Wnd*, der durch den Aufruf von *SetTimer* gegeben ist. Einzelheiten hierzu finden Sie im folgenden Hinweis.

Rückgabewert Der Rückgabewert legt den Integerbezeichner für das neue Timer-Ereignis fest. Ist der Parameter *Wnd* 0, gibt eine Anwendung diesen Wert an die Funktion *KillTimer*, um das Timer-Ereignis zu beseitigen. Der Rückgabewert ist 0, wenn der Timer nicht eingerichtet wurde.

Hinweis Timer sind eine eingeschränkt verfügbare, globale Ressource. Aus diesem Grund ist es wichtig, daß eine Anwendung den durch die Funktion *SetTimer* zurückgegeben Wert überprüft, um sicherzustellen, daß ein Timer verfügbar ist.

Um eine Timer-Funktion zu installieren, muß *SetTimer* eine Adresse der Prozedurinstanz einer Callback-Funktion erhalten und die Funktion muß in der Moduldefinitionsdatei der Anwendung exportiert werden. Die Adresse sollte mit Hilfe der Funktion *MakeProcInstance* erzeugt werden.

Callback-Funktion

```
function TimerFunc(Wnd: HWnd; wMsg: Word; nIDEvent:
Integer; dwTime: LongInt): Word; export;
```

TimerFunc: Platzhalter für den Namen der anwendungsunterstützten Funktion.

hWnd: Bezeichnet das Fenster, das dem Timer-Ereignis zugeordnet wird.

wMsg: Bestimmt die *wm_Timer*-Botschaft.

nIDEvent: Bestimmt die ID des Timers.

dwTime: Bestimmt die aktuelle Systemzeit.

Querverweis *wm_Timer*

SetViewportExt

Syntax

```
function SetViewportExt(DC: HDC; X, Y: Integer):
LongInt;
```

Beschreibung Diese Funktion legt die x- und y-Abmessungen des Grafikfensters des festgelegten Gerätekontextes fest. Das Grafikfenster definiert zusammen mit dem Gerätekontextfenster, wie die GDI Punkte im virtuellen Koordinatensystem auf Punkte im Koordinatensystem des aktuellen Gerätes abbildet.

Die x- und y-Abmessungen des Grafikfensters bestimmen, in welchem Maß die GDI Einheiten im virtuellen Koordinatensystem strecken oder stauchen muß, um sie an die Einheiten im Gerätekoordinatensystem anzupassen.

Ist beispielsweise die x-Abmessung des Fensters 2 und die x-Abmessung des Grafikfensters 4, so überträgt die GDI zwei virtuelle Einheiten (gemessen an der x-Achse) in vier Geräteeinheiten. Ist die y-Abmessung des Fensters 2 und die y-Abmessung des Grafikfensters -1, so überträgt die GDI zwei

virtuelle Einheiten (gemessen an der y-Achse) in eine Geräteeinheit.

Die Größen definieren auch die relative Orientierung der x- und y-Achsen in beiden Koordinatensystemen. Sind die Vorzeichen der zusammengehörenden Fenster- und Grafikfensterabmessungen gleich, haben die Achsen die gleiche Orientierung, sind die Vorzeichen verschieden, so wird die Orientierung umgedreht.

Ist zum Beispiel die y-Abmessung des Fensters 2 und die y-Abmessung des Grafikfensters -1, so stellt die GDI die positive y-Achse im virtuellen Koordinatensystem auf der negativen y-Achse im Gerätekoordinatensystem dar. Sind die x-Abmessungen 2 und 4, stellt die GDI die positive x-Achse im virtuellen Koordinatensystem auf der positiven x-Achse im Gerätekoordinatensystem dar.

DC: Bezeichnet den Gerätekontext.

X: Bestimmt die x-Abmessung des Grafikfensters in Geräteeinheiten.

Y: Bestimmt die y-Abmessung des Grafikfensters in Geräteeinheiten.

Rückgabewert Es werden die vorherigen Abmessungen des Grafikfensters zurückgeliefert. Die vorherige y-Abmessung steht im höherwertigen Word, die vorherige x-Abmessung im niederwertigen. Tritt ein Fehler auf, ist der Rückgabewert 0.

Hinweis Wenn die folgenden Abbildungsmodi gesetzt sind, werden Aufrufe der Funktionen *SetWindowExt* und *SetViewportExt* ignoriert:

mm_HiEnglish
mm_HiMetric
mm_LoEnglish
mm_LoMetric
mm_Text
mm_Twips

Ist der Modus *mm_Isotropic* gesetzt, muß eine Anwendung vor dem Aufruf von *SetViewportExt* die Funktion *SetWindowExt* aufrufen.

SetViewportOrg

Syntax
```
function SetViewportOrg(DC: HDC; X, Y: Integer):
LongInt;
```

Beschreibung Diese Funktion bestimmt den Ursprung des Grafikfensters für den festgelegten Gerätekontext. Das Grafikfenster bestimmt zusammen mit dem Gerätekontextfenster, wie die GDI Punkte

im virtuellen Koordinatensystem auf Punkte im Koordinatensystem des aktuellen Gerätes abbildet.

Der Ursprung des Grafikfensters markiert den Punkt im Gerätekoordinatensystem, auf den die GDI den Ursprung des Fensters abbildet. Es ist ein Punkt im virtuellen Koordinatensystem, der durch die Funktion *SetWindowOrg* bestimmt wird.

Die GDI transformiert alle anderen Punkte mit derselben Methode, die erforderlich ist, um den Ursprung des Fensters auf den Ursprung des Grafikfensters zu übertragen.

So werden beispielsweise alle Punkte eines Kreises um den Ursprung des Fensters in einen Kreis um den Ursprung des Grafikfensters herum angeordnet. Ebenso verlaufen alle Punkte einer Linie, die durch den Ursprung des Fensters geht, in einer Linie durch den Ursprung des Grafikfensters.

DC: Bezeichnet den Gerätekontext.

X: Bestimmt die x-Koordinate des Ursprungs des Grafikfensters in Geräteeinheiten. Der Wert muß innerhalb des Bereichs des Gerätekoordinatensystems liegen.

Y: Bestimmt die y-Koordinate des Ursprungs des Grafikfensters in Geräteeinheiten. Der Wert muß innerhalb des Bereichs des Gerätekoordinatensystems liegen.

Rückgabewert Der Rückgabewert liefert den vorherigen Ursprung des Grafikfensters in Geräteeinheiten zurück.

Die y-Koordinate steht im höherwertigen, die x-Koordinate im niederwertigen Word.

SetVoiceAccent

Syntax
```
function SetVoiceAccent(Voice, Tempo, Volume, Mode,
Pitch: Integer): Integer;
```

Beschreibung Diese Funktion stellt einen festgelegten Ton (Tempo, Lautstärke, Modus und Tonhöhe) in die durch den Parameter *Voice* festgelegte Klang-Warteschlange. Der neue Ton ersetzt den vorherigen und bleibt solange gültig, bis er ersetzt wird. Ein Ton wird nicht als Note gezählt. Wenn in der Warteschlange kein ausreichender Platz ist, tritt ein Fehler auf; *SetVoiceAccent* läßt in der Warteschlange immer Platz für eine einzelne Synchronisationsmarkierung. Liegt *Voice* außerhalb des gültigen Bereichs, wird *SetVoiceAccent* ignoriert.

Voice: Bestimmt eine Klang-Warteschlange. Die erste Warteschlange wird mit 1 numeriert.

Tempo: Bestimmt die Anzahl von Viertelnoten, die pro Minute gespielt werden. Dies kann ein Wert von 32 bis 255 sein, der Standardwert ist 120.

Volume: Bestimmt den Lautstärkepegel. Er kann jeden Wert zwischen 0 (kleinste Lautstärke) und 255 (größte Lautstärke) annehmen.

Mode: Bestimmt, wie die Töne gespielt werden sollen. Er kann einen der folgenden Werte annehmen:

Wert	Bedeutung
s_Legato:	Der Ton wird für die ganze Dauer gehalten und mit dem Beginn des nächsten Tons ausgeblendet.
s_Normal:	Der Ton wird für die ganze Dauer gehalten und bricht ab, bevor der nächste Ton beginnt.
s_Staccato:	Der Ton wird nur für einen Teil der Dauer gehalten und weist einen deutlichen Abstand zwischen ihm und dem nächsten Ton auf.

Pitch: Bestimmt die Tonhöhe der zu spielenden Töne. Dies kann ein Wert zwischen 0 und 83 sein.

Der Tonhöhenwert wird unter Verwendung von Modulo-84-Arithmetik zu jedem Tonwert addiert.

Rückgabewert

Der Rückgabewert ist 0, wenn die Funktion erfolgreich war, ansonsten ist der Rückgabewert einer der folgenden Werte:

Wert	Bedeutung
s_SerDMD	Ungültiger Modus
s_SerDTP	Ungültiges Tempo
s_SerDVL	Ungültige Lautstärke
s_SerQFUL	Warteschlange voll

SetVoiceEnvelope

Syntax

```
function SetVoiceEnvelope(Voice, Shape, RepeatCount:
Integer): Integer;
```

Beschreibung

Diese Funktion reiht die Hüllkurve (Kurvenform und Anzahl der Wiederholungen) in die durch den Parameter *Voice* festgelegte Klang-Warteschlange ein.

Die neue Hüllkurve ersetzt die vorherige und bleibt bis zum nächsten Aufruf von *SetVoiceEnvelope* wirksam. Eine Hüllkurve wird nicht als Ton gezählt.

Wenn in der Warteschlange kein ausreichender Platz ist, tritt ein Fehler auf, *SetVoiceEnvelope* läßt in der Warteschlange immer Platz für eine einzelne Synchronisationsmarkierung. Liegt *Voice* ausserhalb des gültigen Bereichs, wird *SetVoiceEnvelope* ignoriert.

Voice: Bestimmt die Klang-Warteschlange, der die Hüllkurve zugeordnet wird.

Shape: Bestimmt einen Index auf eine OEM-Kurvenformtabelle.

RepeatCount: Bestimmt die Anzahl der Wiederholungen der Kurvenform während der Dauer einer Note.

Rückgabewert Der Rückgabewert ist 0, wenn die Funktion erfolgreich war. Tritt ein Fehler auf, nimmt der Rückgabewert einen der folgenden Werte an:

Wert	Bedeutung
s_SerDRC	Ungültige Anzahl der Wiederholungen
s_SerDSH	Ungültige Kurvenform
s_SerQFUL	Warteschlange voll

SetVoiceNote

Syntax
```
function SetVoiceNote(Voice, Value, Length, Cdots: Integer): Integer;
```

Beschreibung Diese Funktion reiht einen durch die Parameter *Value*, *Length* und *Cdots* bestimmten Ton in die durch den Parameter *Voice* festgelegte Klang-Warteschlange ein.

Ein Fehler tritt auf, wenn in der Warteschlange kein ausreichender Platz ist, die Funktion läßt in der Warteschlange immer Platz für eine einzelne Synchronisationsmarkierung.

Voice: Bestimmt die Klang-Warteschlange, der der Ton zugeordnet wird. Ist *Voice* außerhalb des gültigen Bereichs, wird *SetVoiceNote* ignoriert.

Value: Bestimmt eine von 84 möglichen Noten (sieben Oktaven). Ist *Value* 0, wird eine Pause angenommen.

Length: Bestimmt den Kehrwert der Dauer des Tons. 1 setzt beispielsweise eine ganze Note fest, 2 eine halbe Note, 4 eine viertel Note, usw.

Cdots: Bestimmt die Dauer des Tons in Punkten. Die Dauer ist gleich *Length* * (*Cdots* * 3/2).

Rückgabewert Der Rückgabewert ist 0, wenn die Funktion erfolgreich war. Tritt ein Fehler auf, nimmt der Rückgabewert einen der folgenden Werte an:

Wert	Bedeutung
s_SerDCC	Ungültige Punktanzahl.
s_SerDLN	Ungültige Tonlänge.
s_SerDNT	Ungültige Note.
s_SerQFUL	Warteschlange voll.

SetVoiceQueueSize

Syntax
```
function SetVoiceQueueSize(Voice, Bytes: Integer):
Integer;
```

Beschreibung Diese Funktion reserviert die durch den Parameter *Bytes* festgelegte Anzahl von Bytes für die durch den Parameter *Voice* festgelegte Klang-Warteschlange.

Ist die Größe der Warteschlange nicht bestimmt, ist der Standardwert 192 Bytes, was ungefähr 32 Tönen Platz bietet. Alle Klang-Warteschlangen werden im Speicher gesperrt. Die Warteschlangen können nicht gesetzt werden, während Musik spielt.

Voice: Bestimmt eine Klang-Warteschlange.

Bytes: Bestimmt die Anzahl von Bytes in der Klang-Warteschlange.

Rückgabewert Der Rückgabewert ist 0, wenn die Funktion erfolgreich war. Im Fall eines Fehlers sind folgenden Werte möglich:

Wert	Bedeutung
s_SerMACT	Musik aktiv.
s_SerOFM	zu wenig Speicher.

SetVoiceSound

Syntax
```
function SetVoiceSound(Voice: Integer; Frequency:
LongInt; Duration: Integer): Integer;
```

Beschreibung Diese Funktion stellt Frequenz und Dauer in die durch den Parameter *Voice* festgelegte Klang-Warteschlange.

Voice: Bestimmt eine Klang-Warteschlange. Die erste Klang-Warteschlange wird mit 1 numeriert.

Frequency: Bestimmt die Frequenz. Das höherwertige Word enthält die Frequenz in Hertz, das niederwertige Word enthält die Teilfrequenz.

Duration: Bestimmt die Dauer des Tons in Clock-Ticks.

Rückgabewert Der Rückgabewert ist 0, wenn die Funktion erfolgreich war. Im Fall eines Fehlers sind folgenden Werte möglich:

Wert	Bedeutung
s_SerDDR	Ungültige Dauer.
s_SerDFQ	Ungültige Frequenz.
s_SerDVL	Ungültige Lautstärke.
s_SerQFUL	Warteschlange voll.

SetVoiceThreshold

Syntax
```
function SetVoiceThreshold(Voice, Notes: Integer): Integer;
```

Beschreibung Diese Funktion setzt den Schwellenwert für die durch *Voice* gegebene Klang-Warteschlange.

Wenn die Anzahl der in der Klang-Warteschlange verbleibenden Töne den durch den Parameter *Notes* festgelegten Wert unterschreitet, wird das Schwellenflag gesetzt. Ist beim Aufruf von *SetVoiceTreshold* die Länge der Warteschlange kleiner als *Notes*, wird das Flag nicht gesetzt.

Um den momentanen Schwellenstatus zu ermitteln, sollte die Funktion *GetThresholdStatus* aufgerufen werden.

Voice: Bestimmt die zu setzende Klang-Warteschlange.

Notes: Bestimmt die Anzahl der Töne für den Schwellenwert.

Rückgabewert Der Rückgabewert ist 0, wenn die Funktion erfolgreich war, er ist 1, wenn die Anzahl der Töne in *Notes* außerhalb des gültigen Bereichs liegt.

Querverweis *GetThresholdStatus*

SetWindowExt

Syntax
```
function SetWindowExt(DC: HDC; X, Y: Integer): Longint;
```

Beschreibung Diese Funktion setzt die x- und y-Abmessungen des Fensters, das mit dem vorgegebenen Gerätekontext verbunden ist.

Das Fenster legt zusammen mit dem Gerätekontextfenster fest, wie die GDI Punkte im virtuellen Koordinatensystem auf Punkte im Gerätekoordinatensystem abbildet. Die x- und y-Abmessungen des Fensters bestimmen, in welchem Maß die GDI Einheiten im virtuellen Koordinatensystem strecken oder stauchen muß, um die Einheiten dem Gerätekoordinatensystem anzupassen.

Wenn beispielsweise die x-Abmessung des Fensters 2 und die x-Abmessung des Grafikfensters 4 sind, überträgt die GDI zwei virtuelle Einheiten (gemessen an der x-Achse) in vier Geräteeinheiten. Ist die y-Abmessung des Fensters 2 und die y-Abmessung des Grafikfensters -1, überträgt die GDI zwei virtuelle Einheiten (gemessen an der y-Achse) in eine Geräteeinheit. Die Größen definieren auch die relative Orientierung der x- und y-Achsen in beiden Koordinatensystemen.

Sind die Vorzeichen der zusammengehörenden Fenster- und Grafikfenstergrößen gleich, haben die Achsen die gleiche Orientierung, sind die Vorzeichen verschieden, wird die Orientierung umgedreht. Ist z.B. die y-Abmessung des Fensters 2 und

die y-Abmessung des Grafikfensters -1, so bildet die GDI die positive y-Achse im virtuellen Koordinatensystem auf die negative y-Achse im Gerätekoordinatensystem ab. Sind die x-Abmessungen 2 und 4, bildet die GDI die positive x-Achse im virtuellen Koordinatensystem auf die positive x-Achse im Gerätekoordinatensystem ab.

DC: Bezeichnet den Gerätekontext.

X: Bestimmt die x-Abmessung des Fensters in virtuellen Einheiten.

Y: Bestimmt die y-Abmessung des Fensters in virtuellen Einheiten.

Rückgabewert Der Rückgabewert gibt die vorherigen Abmessungen des Fensters in virtuellen Einheiten an. Die y-Abmessung steht im höherwertigen, die x-Abmessung im niederwertigen Word. Tritt ein Fehler auf, ist der Rückgabewert 0.

Hinweis Wenn die folgenden Abbildungsmodi gesetzt sind, werden Aufrufe einer der Funktionen *SetWindowExt* und *SetViewportExt* ignoriert:

mm_HiEnglish	*mm_LoMetric*
mm_HiMetric	*mm_Text*
mm_LoEnglish	*mm_Twips*

Wenn der Modus *mm_Isotropic* gesetzt ist, muß eine Anwendung die Funktion *SetWindowExt* aufrufen, bevor sie *SetViewportExt* aufruft.

SetWindowLong

Syntax
```
function SetWindowLong(Wnd: HWnd; Index: Integer;
NewLong: LongInt): LongInt;
```

Beschreibung Diese Funktion ändert ein Attribut des durch den Parameter *Wnd* bezeichneten Fensters.

Wnd: Bezeichnet das Fenster.

Index: Bestimmt den Byte-Offset des zu ändernden Attributes. Er kann einen der folgenden Werte annnehmen:

Wert	Bedeutung
gwl_ExStyle	Setzt einen neuen erweiterten Fensterstil.
gwl_Style	Setzt einen neuen Fensterstil.
gwl_WndProc	Setzt einen neuen Zeiger auf die Fensterprozedur.

NewLong: Bestimmt den Austauschwert.

Rückgabewert Der Rückgabewert gibt den vorherigen Wert des angegebenen LongInt an.

Hinweis Wird *SetWindowLong* mit dem Index *gwl_WndProc* verwendet, um eine neue Fensterfunktion zu setzen, muß diese Funktion die Form einer Fensterfunktion haben und in der Moduldefinitionsdatei der Anwendung exportiert werden (siehe die Beschreibung der Funktion *RegisterClass*).

Der Aufruf von *SetWindowLong* mit dem *gcl_WndProc*-Index erzeugt eine Unterklasse der zur Erzeugung des Fensters verwendeten Fensterklasse. Eine Anwendung sollte nicht versuchen, eine Fensterunterklasse wie Kombinationsfenster und Aktionsschalter für Standard-Windows-Steuerelemente zu erzeugen.

Um auf zusätzliche Vier-Byte-Werte zuzugreifen, die beim Erzeugen der Fensterklasse reserviert wurden, verwenden Sie einen positiven Byte-Offset ab dem durch den Parameter *Index* festgelegten Index. Dieser Index ist 0 für den ersten Vier-Byte-Wert im zusätzlichen Speicherbereich, 4 für den nächsten Vier-Byte-Wert, usw.

SetWindowOrg

Syntax
```
function SetWindowOrg(DC: HDC; X, Y: Integer): LongInt;
```

Beschreibung Diese Funktion setzt den Ursprung des Fensters des angegebenen Gerätekontextes fest. Das Fenster legt zusammen mit dem Grafikfenster des Gerätekontextes fest, wie die GDI Punkte im virtuellen Koordinatensystem auf Punkte im Gerätekoordinatensystem abbildet.

Der Ursprung des Fensters markiert den Punkt im virtuellen Koordinatensystem, von dem aus die GDI den Ursprung des Grafikfensters abbildet. Dieser ist ein Punkt im Gerätekoordinatensystem, der durch die Funktion *SetWindowOrg* bestimmt wird. Die GDI transformiert alle anderen Punkte mit derselben Methode, die erforderlich ist, um den Ursprung des Fensters auf den Ursprung des Grafikfensters zu übertragen.

So werden beispielsweise alle Punkte eines Kreises um den Ursprung des Fensters in einem Kreis um den Ursprung des Grafikfensters herum angeordnet. Ebenso verlaufen alle Punkte einer Linie, die durch den Ursprung des Fensters geht, in einer Linie durch den Ursprung des Grafikfensters.

DC: Bezeichnet den Gerätekontext.

X: Legt die virtuelle x-Koordinate des neuen Fensterursprungs fest.

Y: Bestimmt die virtuelle y-Koordinate des neuen Fensterursprungs.

Rückgabewert Der Rückgabewert gibt den vorherigen Ursprung des Fensters an. Die y-Koordinate steht im höherwertigen, die x-Koordinate im niederwertigen Word.

SetWindowPos

Syntax
```
procedure SetWindowPos(Wnd: HWnd; WndInsertAfter: HWnd;
X, Y, cx, cy: Integer; Flags: Word);
```

Beschreibung Diese Prozedur ändert Größe, Position und Reihenfolge von untergeordneten Fenstern, Pop-Up- und Hauptfenstern. Diese Fenster werden in der Reihenfolge ihres Erscheinens am Bildschirm angeordnet, das oberste Fenster erhält den höchsten Rang und ist das erste Fenster in der Liste. Diese Reihenfolge wird in einer Fensterliste aufgezeichnet.

Wnd: Bezeichnet das Fenster, das positioniert wird.

WndInsertAfter: Bezeichnet das Fenster in der Liste des Fenster-Managers, das dem positionierten Fenster vorangeht.

X: Die x-Koordinate der oberen linken Ecke des Fensters.

Y: Die y-Koordinate der oberen linken Ecke des Fensters.

cx: Bestimmt die neue Breite des Fensters.

cy: Bestimmt die neue Höhe des Fensters.

Flags: Bestimmt einen von acht möglichen 16-Bit-Werten, die Größenverhältnisse und Positionierung des Fensters betreffen. Er muß einen der folgenden Werte annehmen:

Wert	Bedeutung
swp_DrawFrame	Zeichnet einen Rahmen (definiert in der Klassenbeschreibung des Fensters) um das Fenster.
swp_HideWindow	Versteckt das Fenster.
swp_NoActivate	Aktiviert das Fenster nicht.
swp_NoMove	Behält die aktuelle Position bei (ignoriert die Parameter *x* und *y*).
swp_NoSize	Behält die aktuelle Größe bei (ignoriert die Parameter *cx* und *cy*).
swp_NoRedraw	Zeichnet Veränderungen nicht neu.
swp_NoZOrder	Behält die aktuelle Reihenfolge bei (ignoriert den Parameter *WndInsertAfter*).
swp_ShowWindow	Zeigt das Fenster.

Hinweis Ist das *swp_NoZOrder*-Flag nicht gesetzt, plaziert Windows das durch den Parameter *Wnd* bezeichnete Fenster an der Position, die dem durch den Parameter *WndInsertAfter* festgelegten Fen-

ster folgt. Ist *WndInsertAfter* 0, so plaziert Windows das durch *Wnd* bestimmte Fenster am Anfang der Liste, ist *WndInsertAfter* 1, plaziert Windows das durch *Wnd* bezeichnete Fenster am Schluß der Liste.

Sind die Flags *swp_ShowWindow* oder *swp_HideWindow* gesetzt, können Bildlauf und Verschieben nicht gleichzeitig erfolgen. Alle Koordinaten für untergeordnete Fenster gelten relativ zur oberen linken Ecke des Client-Bereichs des Hauptfensters.

SetWindowsHook

Syntax

```
function SetWindowsHook(FilterType: Integer;
FilterFunc: TFarProc): TFarProc;
```

Beschreibung

Diese Funktion installiert eine Filterfunktion in einer Kette. Eine Filterfunktion bearbeitet Ereignisse, bevor diese an die Botschaftschleife in der Funktion *WinMain* einer Anwendung gesendet werden. Eine Kette ist eine verkettete Liste von Filterfunktionen desselben Typs.

FilterType: Bestimmt die zu installierende Systemfilterverwaltung. Er kann einen der folgenden Werte annehmen:

Wert	Bedeutung
wh_CallWndProc	Installiert einen Fensterfunktionsfilter.
wh_GetMessage	Installiert einen Botschaftsfilter.
wh_JournalPlayback	Installiert einen Wiedergabeprotokollfilter.
wh_JournalRecord	Installiert einen Aufzeichnungsprotokollfilter.
wh_Keyboard	Installiert einen Tastaturfilter.
wh_MsgFilter	Installiert einen Botschaftsfilter.
wh_SysMsgFilter	Installiert einen systemweiten Botschaftsfilter.

FilterFunc: Ist die Adresse der Prozedurinstanz der zu installierenden Filterfunktion. Einzelheiten finden Sie im folgenden Hinweis.

Rückgabewert

Der Rückgabewert zeigt auf die Adresse der Prozedurinstanz des vorher installierten Filters. Er ist 0, wenn kein vorheriger Filter existiert. Die Anwendung oder Bibliothek, die die Funktion *SetWindowsHook* aufruft, sollte den Rückgabewert im Datensegment der Bibliothek sichern. Der vierte Parameter der Funktion *DefHookProc* zeigt auf die Position im Speicher, an der die Bibliothek diesen Rückgabewert sichert.

Hinweis

Die *wh_CallWndProc*-Filterverwaltung beeinflußt die Leistung des Systems. Sie ist nur zum Testen von Anwendungen vorgesehen.

Die Systemfilterverwaltungen sind gemeinsam genutzte Ressourcen. Die Installation einer Filterverwaltung beeinflußt alle Anwendungen. Die meisten Filterverwaltungsfunktionen müssen in Bibliotheken installiert werden, ausgenommen *wh_MsgFilter,* die task-spezifisch ist. Systemfilterverwaltungen sollten auf Sonderanwendungen beschränkt werden oder als Entwicklungshilfsmittel während des Testens einer Anwendung eingesetzt werden. Bibliotheken, die die Filterverwaltung nicht mehr benötigen, sollten die Filterfunktion entfernen.

Um eine Filterfunktion zu installieren, muß *SetWindowsHook* die Adresse der Prozedurinstanz der Funktion erhalten und die Funktion muß in der Moduldefinitionsdatei der Bibliothek exportiert werden. Bibliotheken können die Funktionsadresse direkt übergeben. Tasks müssen *MakeProcInstance* verwenden, um die Adresse abzurufen. DLLs müssen *GetProcAddress* verwenden, um die Adresse zu erhalten.

Der folgende Abschnitt beschreibt, wie die einzelnen Filterverwaltungsfunktionen unterstützt werden.

wh_CallWndProcFilterfunktion

Windows ruft die *wh_CallWndProc*-Filterfunktion immer dann auf, wenn die Funktion *SendMessage* aufgerufen wird (aber nicht, wenn die Funktion *PostMessage* aufgerufen wird).

```
procedure  FilterFunc (nCode: Integer; wParam: Word;
lParam: LongInt);    export ;
```

FilterFunc: Platzhalter für den anwendungs- oder bibliotheksunterstützten Funktionsnamen.

nCode: Bestimmt, ob die Filterfunktion die Botschaft bearbeiten oder die Funktion *DefHookProc* aufrufen soll. Ist der Parameter *nCode* kleiner als 0, sollte die Filterfunktion die Botschaft ohne weitere Bearbeitung an *DefHookProc* übergeben.

wParam: Bestimmt, ob die Botschaft von der aktuellen Task gesendet wird. Er ist ungleich 0, wenn dies der Fall ist, ansonsten 0.

lParam: Zeigt auf eine Datenstruktur, die Einzelheiten über die durch den Filter abgefangene Botschaft enthält.

Nachstehend werden die Reihenfolge, der Typ und die Beschreibung eines jeden Feldes der Datenstruktur aufgeführt:

Feld	Typ/Beschreibung
hlParam	Enthält das höherwertige Word des Parameters *lParam* der durch den Filter erhaltenen Botschaft.
llParam	Enthält das niederwertige Word des Parameters *lParam* der durch den Filter erhaltenen Botschaft.
wParam	Enthält den Parameter *wParam* der durch den Filter erhaltenen Botschaft.

Feld	Typ/Beschreibung
wMsg	Enthält die durch den Filter erhaltene Botschaft.
hWnd	Enthält das Handle des Fensters, das die Botschaft erhalten soll.

Hinweis

Die *wh_CallWndProc*-Filterfunktion kann die Botschaft nach Wunsch prüfen oder modifizieren. Ist die Steuerung an Windows zurückgegeben, wird die Botschaft mit allen Modifikationen zur Fensterfunktion hin übergeben.

Die Filterfunktion hat keinen Rückgabewert.

wh_GetMessage-Filterfunktion

Windows ruft die *wh_GetMessage*-Filterfunktion immer dann auf, wenn die Funktion *GetMessage* aufgerufen wird, sofort nachdem *GetMessage* eine Botschaft von einer Anwendungs-Warteschlange erhalten hat.

```
procedure  FilterFunc (nCode: Integer; wParam: Word;
lParam: LongInt); LongInt;      export ;
```

FilterFunc: Platzhalter für den bibliotheksunterstützten Funktionsnamen.

nCode: Bestimmt, ob die Filterfunktion die Botschaft bearbeiten oder die Funktion *DefHookProc* aufrufen soll. Ist der Parameter *nCode* kleiner als 0, sollte die Filterfunktion die Botschaft ohne weitere Bearbeitung an *DefHookProc* weitergeben.

wParam: Legt einen Null-Wert fest.

lParam: Zeigt auf eine Botschaftsstruktur.

Hinweise

Die *wh_GetMessage*-Filterfunktion kann die Botschaft nach Wunsch prüfen oder modifizieren. Wird die Steuerung dann an Windows zurückgegeben, gibt die Funktion *GetMessage* die Botschaft mit allen Modifikationen an die Anwendung zurück, die sie ursprünglich aufgerufen hat.

wh_Journal-Playback-Filterfunktion

Windows ruft die *wh_JournalPlayback*-Filterfunktion immer dann auf, wenn eine Anfrage für eine Ereignisbotschaft anliegt. Die Funktion ist dafür gedacht, eine vorher aufgezeichnete Ereignisbotschaft bereitzustellen.

```
function  FilterFunc (nCode: Integer; wParam: Word;
lParam: LongInt); LongInt;      export ;
```

FilterFunc: Platzhalter für den bibliotheksunterstützten Funktionsnamen.

nCode: Bestimmt, ob die Filterfunktion die Botschaft bearbeiten oder die Funktion *DefHookProc* aufrufen soll. Ist der Parameter *nCode* kleiner als 0, sollte die Filterfunktion die Botschaft ohne weitere Bearbeitung an *DefHookProc* weitergeben.

wParam: Legt einen Null-Wert fest.

lParam: Zeigt auf die Botschaft, die durch die Filterfunktion bearbeitet wird.

Hinweis

Die *wh_JournalPlayback*-Funktion sollte eine Ereignisbotschaft auf den Parameter *lParam* kopieren. Die Botschaft muß vorher durch den *wh_JournalRecord*-Filter aufgezeichnet worden sein, dieser darf die Botschaft nicht modifizieren.

Während die *wh_JournalPlayback*-Funktion wirksam ist, ignoriert Windows alle Maus- und Tastatureingaben.

Rückgabewert

Der Rückgabewert sollte die Zeitdauer (in Clock-Ticks) sein, die Windows warten soll, bevor die Botschaft bearbeitet wird. Diese Zeit kann berechnet werden, indem die Differenz zwischen den Zeitfeldern in der aktuellen und der vorherigen Ereignisbotschaft gebildet wird. Gibt die Funktion 0 zurück, wird die Botschaft unverzüglich bearbeitet – wird die Steuerung dann an Windows zurückgegeben, wird die Botschaft weiter bearbeitet.

Wenn der Parameter *nCode Hc_Skip* ist, sollte die Filterfunktion die nächste aufgezeichnete Ereignisbotschaft für die Rückgabe bei ihrem nächsten Aufruf vorbereiten.

wh_Journal-Record-Filterfunktion

Windows ruft die *wh_JournalRecord*-Filterfunktion immer dann auf, wenn sie eine Ereignisbotschaft aus der Ereignis-Warteschlange bearbeitet. Der Filter kann dazu verwendet werden, um das Ereignis für eine spätere Wiedergabe aufzuzeichnen.

```
procedure  FilterFunc (nCode: Integer; wParam: Word;
lParam: LongInt);    export ;
```

FilterFunc: Platzhalter für den bibliotheksunterstützten Funktionsnamen.

nCode: Bestimmt, ob die Filterfunktion die Botschaft bearbeiten oder die Funktion *DefHookProc* aufrufen soll. Ist der Parameter *nCode* kleiner als 0, sollte die Filterfunktion die Botschaft ohne weitere Bearbeitung an *DefHookProc* weitergeben.

wParam: Legt einen Null-Wert fest.

lParam: Zeigt auf eine Botschaftsstruktur.

Hinweise

Die *wh_JournalRecord*-Filterfunktion sollte eine Kopie der Ereignisbotschaft für eine spätere Wiedergabe anlegen und die Botschaft nicht modifizieren. Wird die Steuerung dann an Windows zurückgegeben, wird die Botschaft weiter bearbeitet.

wh_Keyboard-Filterfunktion

Windows ruft die *wh_Keyboard*-Filterfunktion immer dann auf, wenn eine Anwendung die Funktionen *GetMessage* oder *PeekMessage* aufruft und ein Tastaturereignis (*wm_KeyUp* oder *wm_KeyDown*) zu bearbeiten ist.

```
function  FilterFunc (nCode: Integer; wParam: Word;
lParam: LongInt); Integer; export;
```

FilterFunc: Platzhalter für den bibliotheksunterstützten Funktionsnamen.

nCode: Bestimmt, ob die Filterfunktion die Botschaft bearbeiten oder die Funktion *DefHookProc* aufrufen soll. Ist dieser Wert *hc_NoRemove*, verwendet die Anwendung die Funktion *PeekMessage* mit der Option *pm_NoRemove* und entfernt die Botschaft nicht aus der System-Warteschlange. Ist dieser Wert kleiner als 0, sollte die Filterfunktion die Botschaft ohne weitere Bearbeitung an *DefHookProc* weitergeben.

wParam: Bestimmt den virtuellen Tastencode der gegebenen Taste.

lParam: Legt die Anzahl der Wiederholungen, den Scan-Code, den Code für den Tastenzustand, den vorherigen Tastenstatus und den Kontextcode wie folgt fest: Bit 0 ist das niederwertige Bit:

Bit	Wert
0-15	(niederwertiges Word) Anzahl der Wiederholungen (wieviele Male ein Tastendruck wiederholt wird, wenn der Anwender die Taste gedrückt hält).
16-23	(niederwertiges Byte des höherwertigen Word) Scan-Code (OEM-abhängiger Wert).
24	Funktionstaste (1, wenn es eine Funktionstaste ist).
25-26	Nicht verwendet. 27-28 (Kontextcode) 1, wenn die Taste Alt gedrückt war, als die Taste betätigt wurde, 0 in anderen Fällen). Wird von Windows intern verwendet.
30	Vorheriger Tastenstatus (1, wenn die Taste gedrückt war, bevor die Botschaft gesendet wurde, 0, wenn die Taste nicht gedrückt war).
31	Tastenzustand (1, wenn die Taste losgelassen wird, 0, wenn die Taste gedrückt ist).

Rückgabewert

Der Rückgabewert gibt an, was mit der Botschaft geschehen soll. Er ist 0, wenn die Botschaft von Windows bearbeitet werden soll und 1, wenn die Botschaft verworfen werden sollte.

wh_MsgFilter-Filterfunktion

Windows ruft die *wh_MsgFilter*-Filterfunktion immer dann auf, wenn ein Dialogfenster, ein Meldungsfenster oder ein Menü eine Botschaft erhalten hat, und bevor diese Botschaft bearbeitet wird. Der Filter erlaubt einer Anwendung, die Botschaften zu bearbeiten oder zu modifizieren.

Anmerkung: Dies ist der einzige Task-spezifische Filter. Eine Task kann diesen Filter installieren.

```
function  FilterFunc (nCode: Integer; wParam: Word;
lParam: LongInt); Integer; export;
```

FilterFunc: Platzhalter für den bibliotheks- oder anwendungsunterstützten Funktionsnamen.

nCode: Bestimmt den Typ der bearbeiteten Botschaft. Er muß einen der folgenden Werte annehmen:

Wert	Bedeutung
msgf_DialogBox	Bearbeitet Botschaften innerhalb einer Dialogfenster- oder einer Meldungsfensterfunktion.
msgf_Menu	Bearbeitet Tastatur- und Mausbotschaften in einem Menü. Ist dieser Wert kleiner als 0, sollte die Filterfunktion die Botschaft ohne weitere Bearbeitung an *DefHookProc* weitergeben.

wParam: Legt einen Null-Wert fest.

lParam: Zeigt auf die Botschaftsstruktur.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn die Filterverwaltungsfunktion die Botschaft bearbeitet, andernfalls ist er 0.

wh_SysMsgFilter-Filterfunktion

Windows ruft die *wh_SysMsgFilter*-Filterfunktion immer dann auf, wenn ein Dialogfenster, ein Meldungsfenster oder ein Menü eine Botschaft erhalten haben und bevor diese Botschaft bearbeitet wurde. Der Filter erlaubt einer Anwendung, Botschaften für jede beliebige Anwendung des Systems zu bearbeiten oder zu modifizieren.

```
function  FilterFunc (nCode: Integer; wParam: Word;
lParam: LongInt); Integer; export;
```

FilterFunc ist ein Platzhalter für den bibliotheksunterstützten Funktionsnamen.

nCode: Bestimmt den Typ der bearbeiteten Botschaft. Er kann einen der folgenden Werte annehmen:

Wert	Bedeutung
msgf_DialogBox	Bearbeiten von Botschaften innerhalb der Funktion *DialogBox*.
msgf_Menu	Bearbeiten von Tastatur- und Mausbotschaften im Menü.
msgf_MessageBox	Bearbeiten von Botschaften innerhalb der Funktion *MessageBox*. Ist dieser Wert kleiner als 0, sollte die Filterfunktion die Botschaft ohne weitere Bearbeitung an *DefHookProc* weitergeben.

wParam: Legt einen Null-Wert fest.

lParam: Zeigt auf die Botschaftsstruktur.

Rückgabewert Der Rückgabewert ist ungleich 0, wenn die Filterverwaltungsfunktion die Botschaft bearbeitet, andernfalls ist er 0.

Querverweis *DefHookProc*

SetWindowText

Syntax
```
procedure SetWindowText(Wnd: HWnd; Str: PChar);
```

Beschreibung Diese Prozedur setzt die Überschrift des gegebenen Fensters auf den String, auf den der Parameter *Str* zeigt. Ist der Parameter *Wnd* ein Handle auf ein Steuerelement, setzt die Prozedur *SetWindowText* den Text innerhalb des Steuerelements.

Wnd: Bezeichnet das Fenster oder das Steuerelement, dessen Text geändert werden soll.

Str: Zeigt auf einen nullterminierten String.

SetWindowWord

Syntax
```
function SetWindowWord(Wnd: HWnd; Index: Integer;
NewWord: Word): Word;
```

Beschreibung Diese Funktion ändert das Attribut eines durch den Parameter *Wnd* bezeichneten Fensters.

Wnd: Bezeichnet das zu modifizierende Fenster.

Index: Bestimmt den Byte-Offset des zu ändernden Word. Er kann auch einen der folgenden Werte annehmen:

Wert	Bedeutung
gww_HInstance	Instanz-Handle des Moduls, welches das Fenster besitzt.
gww_ID	Steuerelement des untergeordneten Fensters.

NewWord: Bestimmt den Austauschwert.

Rückgabewert Der Rückgabewert gibt den vorherigen Wert des festgelegten Word an.

Hinweis Verwenden Sie einen positiven Byte-Offset als Index, der durch den Parameter *Index* festgelegt ist, um auf zusätzliche Zwei-Byte-Werte zuzugreifen, die reserviert wurden, als die Fensterklassenstruktur erzeugt wurde. Der Byte-Offset ist 0 für den ersten Zwei-Byte-Wert im zusätzlichen Speicher, 2 für den nächsten Zwei-Byte-Wert, usw.

ShowCaret

Syntax
```
procedure ShowCaret(Wnd: HWnd);
```

Beschreibung Diese Prozedur zeigt das Caret auf dem Bildschirm an der aktuellen Position an. Sobald es sichtbar ist, beginnt das Caret automatisch zu blinken.

ShowCaret zeigt das Caret nur, wenn es eine aktuelle Form hat und es nicht zweimal oder öfter in Folge verborgen wurde. Besitzt das gegebene Fenster das Caret nicht, wird es nicht angezeigt. Ist der Parameter *Wnd* 0, zeigt die Prozedur *ShowCaret* das Caret nur, wenn es zu einem Fenster der aktuellen Task gehört.

Das Verbergen des Caret geschieht kumulativ. Wurde die Prozedur *HideCaret* fünfmal in Folge aufgerufen, muß *ShowCaret* fünfmal aufgerufen werden, um das Caret sichtbar zu machen.

Wnd: Bezeichnet das Fenster, zu dem das Caret gehört, oder ist 0, um indirekt das Besitzerfenster in der aktuellen Task anzugeben.

Hinweis Das Caret ist eine gemeinsam genutzte Ressource.

Ein Fenster sollte das Caret nur zeigen, wenn es den Fokus hat oder aktiv ist.

ShowCursor

Syntax
```
function ShowCursor(Show: Bool): Integer;
```

Beschreibung Diese Funktion zeigt oder verbirgt den Mauszeiger. *ShowCursor* setzt dazu einen internen Anzeigezähler, der bestimmt, ob der Mauszeiger gezeigt werden soll. Ist der Parameter *Show True*, wird der Anzeigezählerstand um 1 erhöht.

Der Mauszeiger wird nur angezeigt, wenn der Anzeigezählerstand größer als oder gleich 0 ist. Zu Beginn steht der Anzeigezählerstand auf 0, wenn eine Maus installiert ist, ist dies nicht der Fall, steht er auf -1.

Show: Bestimmt, ob der Anzeigezählerstand erhöht oder verringert wird. Der Anzeigezählerstand wird erhöht, wenn *Show True* ist, andernfalls wird er verringert.

Rückgabewert Der Rückgabewert gibt den neuen Anzeigezählerstand an.

Hinweis Der Mauszeiger ist eine gemeinsam genutzte Ressource. Ein Fenster, das den Mauszeiger verbirgt, sollte den Mauszeiger anzeigen, bevor der Mauszeiger seinen Client-Bereich verläßt oder bevor das Fenster die Steuerung an ein anderes Fenster abgibt.

ShowOwnedPopups

Syntax
```
procedure ShowOwnedPopups(Wnd: HWnd; Show: Bool);
```

Beschreibung Diese Prozedur zeigt oder verbirgt alle Pop-Up-Fenster, die zu dem durch den Parameter *Wnd* bezeichneten Fenster gehören. Ist der Parameter *Show True*, werden alle verborgenen Pop-up-Fenster gezeigt, ist er *False* , werden alle sichtbaren Pop-up-Fenster verborgen.

Wnd: Bestimmt das Fenster, zu dem die Pop-up-Fenster gehören, die gezeigt oder verborgen werden sollen.

Show: Bestimmt, ob Pop-up-Fenster versteckt werden oder nicht. Er ist *True*, wenn alle verborgenen Pop-up-Fenster gezeigt werden sollen und *False*, wenn alle sichtbaren Pop-up-Fenster verborgen werden sollen.

ShowScrollBar

Syntax
```
procedure ShowScrollBar(Wnd: HWnd; Bar: Word; Show:
Bool);
```

Beschreibung Diese Prozedur zeigt oder verbirgt eine Bildlaufleiste, abhängig vom Wert des Parameters *Show*. Ist *Show True*, wird die Bildlaufleiste angezeigt, ist *Show False*, wird die Bildlaufleiste verborgen.

Wnd: Bezeichnet ein Fenster, das in seinem Nicht-Client-Bereich eine Bildlaufleiste enthält, wenn der Parameter *Bar sb_Horz, sb_Vert,* oder *sb_Both* ist. Ist *Bar sb_CTL*, legt *Wnd* eine selbständige Bildlaufleiste fest.

Bar: Bestimmt, ob die Bildlaufleiste ein Steuerelement oder Teil des Nicht-Client-Bereichs eines Fensters ist. Ist sie Teil des Nicht-Client-Bereichs, zeigt *Bar* auch an, ob die horizontale, die vertikale oder beide Bildlaufleiste(n) betroffen sind.

Dieser Parameter muß einen der folgenden Werte annehmen:

Wert	Bedeutung
sb_Both	Bestimmt die horizontale und vertikale Bildlaufleiste des Fensters.
sb_CTL	Bestimmt, daß die Bildlaufleiste ein Steuerelement ist.
sb_Horz	Bestimmt die horizontale Bildlaufleiste des Fensters.
sb_Vert	Bestimmt die vertikale Bildlaufleiste des Fensters.

Show: Bestimmt, ob Windows die Bildlaufleiste verbirgt oder nicht. Ist *Show False*, wird die Bildlaufleiste verborgen, andernfalls wird die Bildlaufleiste angezeigt.

Hinweis Eine Anwendung sollte diese Funktion nicht zum Ausblenden einer Bildlaufleiste verwenden, wenn sie gerade eine Bildlaufleistenbotschaft bearbeitet.

ShowWindow

Syntax
```
function ShowWindow(Wnd: HWnd; CmdShow: Integer): Bool;
```

Beschreibung Diese Funktion zeigt oder entfernt das gegebene Fenster, wie es durch den Parameter *CmdShow* festgelegt ist.

Wnd: Bezeichnet das Fenster.

CmdShow: Bestimmt, wie das Fenster gezeigt werden soll. Er muß einen der Werte annehmen, die im Hinweis beschrieben sind.

Rückgabewert Der Rückgabewert ist *True* , wenn das Fenster vorher sichtbar war und *False*, wenn das Fenster vorher verborgen war.

Hinweis Die Funktion *ShowWindow* darf pro Programm nur einmal mit dem Parameter *CmdShow* aus der Funktion *WinMain* aufgerufen werden. Spätere Aufrufe von *ShowWindow* müssen einen in der folgenden Tabelle aufgeführten Werte verwenden. Folgende Werte sind für den Parameter *CmdShow* möglich:

Wert	Bedeutung
sw_Hide	Verbirgt das Fenster und gibt die Aktivierung an ein anderes Fenster ab.
sw_Minimize	Verkleinert das festgelegte Fenster und aktiviert das Hauptfenster in der Liste des Fenster-Managers.
sw_Restore	Wie *sw_ShowNormal*.
sw_Show	Aktiviert ein Fenster und zeigt es in seiner aktuellen Größe und Position.
sw_ShowMaximized	Aktiviert das Fenster und zeigt es als ein vergrößertes Fenster.
sw_ShowMinimized	Aktiviert ein Fenster und zeigt es als Symbol.
sw_ShowMinNoActive	Zeigt das Fenster als Symbol. Das gegenwärtig aktive Fenster bleibt aktiv.
sw_ShowNA	Zeigt das Fenster in seinem aktuellen Zustand. Das gegenwärtig aktive Fenster bleibt aktiv.
sw_ShowNoActivate	Zeigt ein Fenster in seiner neuesten Größe und Position. Das gegenwärtig aktive Fenster bleibt aktiv.
sw_ShowNormal	Aktiviert ein Fenster und zeigt es an. Ist das Fenster verkleinert oder vergrößert, stellt Windows es in seiner ursprünglichen Größe und Position wieder her.

SizeofResource

Syntax
```
function SizeofResource(Instance, ResInfo: THandle):
Word;
```

Beschreibung Diese Funktion liefert die Größe der angegebenen Ressource in Bytes. Sie wird üblicherweise mit der Funktion *AccessResource* verwendet, um Speicher vorzubereiten, der eine Ressource aus einer Datei aufnehmen sollte.

Instance: Bezeichnet die Instanz des Moduls, dessen ausführbare Datei die Ressource enthält.

ResInfo: Bezeichnet die gewünschte Ressource. Von diesem Handle wird angenommen, daß es durch Anwendung der Funktion *FindResource* erzeugt wurde.

Rückgabewert Der Rückgabewert gibt die Anzahl der Bytes in der Ressource an. Er ist 0, wenn die Ressource nicht gefunden werden kann.

Hinweis Der zurückgegebene Wert kann wegen ihrer Ausrichtung grösser sein als die aktuelle Ressource selbst. Eine Anwendung sollte sich nicht auf diesen Wert als exakte Größe einer Ressource stützen.

Querverweis *AccessResource*

StartSound

Syntax
```
function StartSound: Integer;
```

Beschreibung Diese Funktion startet die Wiedergabe in jeder Klang-Warteschlange.

Die Funktion *StartSound* ist nicht destruktiv, sie kann also beliebig oft aufgerufen werden, um die aktuellen Klang-Warteschlangen wiederzugeben.

Diese Funktion hat keine Parameter.

Rückgabewert Obwohl der Rückgabewert vom Typ *Integer* ist, sollte sein Inhalt ignoriert werden.

StopSound

Syntax
```
function StopSound: Integer;
```

Beschreibung Diese Funktion beendet die Wiedergabe aller Klang-Warteschlangen und löscht dann die Inhalte der Warteschlangen. Der Sound-Treiber jeder Stimme wird abgeschaltet.

Diese Funktion hat keine Parameter.

Rückgabewert Obwohl der Rückgabewert vom Typ *Integer* ist, sollte sein Inhalt ignoriert werden.

StretchBlt

Syntax

```
function StretchBlt(DestDC: HDC; X, Y, Width, Height:
Integer; SrcDC: HDC; XSrc, YSrc, SrcWidth, SrcHeight:
Integer; Rop: Longint): Bool;
```

Beschreibung

Diese Funktion bewegt ein Bitmap von einem Quellrechteck in ein Zielrechteck und streckt oder staucht das Bitmap – falls nötig – um seine Abmessungen dem Zielrechteck anzupassen. *StretchBlt* verwendet den Streckmodus des Zielgerätekontextes (gesetzt durch die Funktion *SetStretchBltMode*) um zu bestimmen, wie das Bitmap gestreckt oder gestaucht werden muß.

StretchBlt bewegt das Bitmap von dem durch den Parameter *SrcDC* bezeichneten Quellgerät zu dem durch den Parameter *DestDC* angegebenen Zielgerät. Die vier Parameter *XSrc*, *YSrc*, *SrcWidth* und *SrcHeight* bestimmen den Ursprung und die Abmessungen des Quellrechtecks, die Parameter *X*, *Y*, *Width* und *Height* geben den Ursprung und die Abmessungen des Zielrechtecks an. Die durch den Parameter *Rop* festgelegte Rasteroperation bestimmt, wie das Quell-Bitmap und die auf dem Zielgerät vorhandenen Bits kombiniert werden.

StretchBlt erzeugt ein Spiegelbild des Bitmaps, wenn die Vorzeichen der Parameter *SrcWidth* und *Width* oder *SrcHeight* und *Height* voneinander abweichen. Wenn *SrcWidth* und *Width* unterschiedliche Vorzeichen haben, erzeugt die Funktion ein Spiegelbild des Bitmaps entlang der x-Achse. Haben *SrcHeight* und *Height* unterschiedliche Vorzeichen, erzeugt die Funktion ein Spiegelbild des Bitmaps entlang der y-Achse.

DestDC: Bezeichnet den Gerätekontext, der das Bitmap aufnehmen soll.

X: Bestimmt die virtuelle x-Koordinate der oberen linken Ecke des Zielrechtecks.

Y: Bestimmt die virtuelle y-Koordinate der oberen linken Ecke des Zielrechtecks.

Width: Bestimmt die Breite des Zielrechtecks (in virtuellen Einheiten).

Height: Bestimmt die Höhe des Zielrechtecks (in virtuellen Einheiten).

SrcDC: Bezeichnet den Gerätekontext, der das Quell-Bitmap enthält.

XSrc: Bestimmt die virtuelle x-Koordinate der oberen linken Ecke des Quellrechtecks.

YSrc: Bestimmt die virtuelle y-Koordinate der oberen linken Ecke des Quellrechtecks.

SrcWidth: Bestimmt die Breite des Quellrechtecks (in virtuellen Einheiten).

SrcHeight: Bestimmt die Höhe des Quellrechtecks (in virtuellen Einheiten).

Rop: Legt die durchzuführende Rasteroperation fest. Die Codes der Rasteroperationen bestimmen, wie die GDI Farben bei Ausgabeoperationen kombiniert, die einen aktiven Pinsel, ein mögliches Quell-Bitmap und ein Ziel-Bitmap einschließen. Eine Liste der Codes für Rasteroperationen finden Sie in Kapitel 1 »Windows-Stile und -Konstanten« im Abschnitt »Ternäre Rasteroperationen«.

Rückgabewert Der Rückgabewert gibt an, ob das Bitmap gezeichnet wurde. Er ist *True*, wenn das Bitmap gezeichnet wurde, andernfalls ist er *False*.

Hinweis *StretchBlt* streckt oder staucht das Quell-Bitmap im Speicher und kopiert dann das Ergebnis in das Ziel. Soll ein Muster mit dem Ergebnis verschmolzen werden, wird es solange nicht verschmolzen, bis das gestreckte Quell-Bitmap in das Ziel kopiert wird. Wird ein Pinsel verwendet, ist es der angewählte Pinsel im Zielgerätekontext.

Die Zielkoordinaten werden in Übereinstimmung mit dem Zielgerätekontext, die Quellkoordinaten in Übereinstimmung mit dem Quellgerätekontext umgewandelt.

Wenn Ziel-, Quell- und Muster-Bitmaps nicht das gleiche Farbformat haben, konvertiert *StretchBlt* das Quell- und das Muster-Bitmap so, daß sie zum Ziel-Bitmap passen. Die Vorder- und Hintergrundfarben des Ziels werden bei der Konversion verwendet.

Wenn *StretchBlt* ein Monochrom-Bitmap in Farbe konvertieren muß, werden weiße Bits (1) auf die Hintergrundfarbe und schwarze Bits (0) auf die Vordergrundfarbe gesetzt. Um Farbe in Monochrom zu konvertieren, werden Pixel, die der Hintergrundfarbe entsprechen, auf weiß (1) und alle anderen Pixel auf schwarz (0) gesetzt. Bei Farbgeräten werden stattdessen die Vorder- und Hintergrundfarben des Gerätekontextes verwendet.

Nicht alle Geräte unterstützen die Funktion *StretchBlt*.

Querverweis *SetStretchBltMode*

StretchDIBits

Syntax

```
function StretchDIBits(DC: HDC; DestX, DestY,
DestWidth, DestHeight, SrcX, SrcY, SrcWidth, SrcHeight:
Word; Bits: Pointer; var BitsInfo: TBitmapInfo; Usage:
Word; Rop: LongInt): Integer;
```

Beschreibung Diese Funktion bewegt ein geräteunbhängiges Bitmap (device-independent bitmap, DIB) aus einem Quellrechteck in ein Zielrechteck und streckt oder staucht das Bitmap, wenn dies nötig ist, um es den Abmessungen des Zielrechtecks anzupassen.

StretchDIBits verwendet den Streckmodus des Zielgerätekontextes (gesetzt durch die Funktion *SetStretchBltMode*), um zu bestimmen, wie das Bitmap zu strecken oder zu stauchen ist.

StretchDIBits bewegt das Bitmap von dem durch die Parameter *Bits*, *BitsInfo* und *Usage* festgelegten geräteunabhängigen Bitmap zu dem durch den Parameter *DC* bezeichneten Zielgerät. Die Parameter *XSrc*, *YSrc*, *SrcWidth* und *SrcHeight* definieren den Ursprung und die Abmessungen des Quellrechtecks. Der Ursprung des Koordinatensystems des geräteunabhängigen Bitmaps ist die untere linke Ecke. Die Parameter *DestX*, *DestY*, *DestWidth* und *DestHeight* geben den Ursprung und die Abmessungen des Zielrechtecks an. Der Koordinatenursprung des Ziels hängt vom aktuellen Abbildungsmodus des Gerätekontextes ab. Weitere Informationen zu den Abbildungsmodi finden Sie im Abschnitt »mm_Xxxx« in Kapitel 1 »Windows-Stile und -Konstanten«.

Die durch den Parameter *Rop* festgelegte Rasteroperation bestimmt, wie das Quell-Bitmap und die gegebenen Bits des Zielgerätes kombiniert werden.

StretchBlt erzeugt ein Spiegelbild des Bitmaps, wenn die Vorzeichen der Parameter *SrcWidth* und *DestWidth* oder *SrcHeight* und *DestHeight* voneinander abweichen.

Haben *SrcWidth* und *DestWidth* unterschiedliche Vorzeichen, erzeugt die Funktion ein Spiegelbild des Bitmaps entlang der x-Achse, haben *SrcHeight* und *DestHeight* unterschiedliche Vorzeichen, erzeugt die Funktion ein Spiegelbild des Bitmaps entlang der y-Achse.

DC: Bezeichnet den Zielgerätekontext für eine Bildschirmoberfläche oder ein Speicher-Bitmap.

DestX: Bestimmt die x-Koordinate des Ursprungs des Zielrechtecks (in virtuellen Einheiten).

DestY: Bestimmt die y-Koordinate des Ursprungs des Zielrechtecks (in virtuellen Einheiten).

DestWidth: Bestimmt die Breite des Zielrechtecks (in virtuellen Einheiten).

DestHeight: Bestimmt die Höhe des Zielrechtecks (in virtuellen Einheiten).

SrcX: Bestimmt die x-Koordinate des Quellrechtecks im DIB in Pixel.

SrcY: Bestimmt die y-Koordinate des Quellrechecks im DIB in Pixel.

SrcWidth: Bestimmt die Breite des Quellrechtecks im DIB in Pixel.

SrcHeight: Bestimmt die Höhe des Quellrechtecks im DIB in Pixel.

Bits: Bits des DIB, die in einem Array von Bytes abgelegt sind.

BitsInfo: Eine *TBitmapInfo*-Datenstruktur, die Informationen über das DIB enthält.

Usage: Bestimmt, ob die Felder *bmiColors[ ]* in *BitsInfo* explizite RGB-Werte oder Indizes auf die gegenwärtig verwendete virtuelle Palette enthalten. Dieser Parameter muß einen der folgenden Werte annehmen:

Wert	Bedeutung
DIB_Pal_Colors	Die Farbtabelle besteht aus einem Array von 16-Bit-Indizes auf die aktuell verwendete virtuelle Palette.
DIB_RGB_Colors	Die Farbtabelle enthält RGB-Werte.

Rop: Legt die durchzuführende Rasteroperation fest. Die Codes der Rasteroperationen bestimmen, wie die GDI Farben bei Ausgabeoperationen kombiniert, die einen aktiven Pinsel, ein mögliches Quell-Bitmap und ein Ziel-Bitmap einschließen.

Weitere Informationen finden Sie in Kapitel 1 »Windows-Stile und -Konstanten« im Abschnitt »Ternäre Rasteroperationen«.

Rückgabewert Der Rückgabewert ist die Anzahl der kopierten Scan-Zeilen.

SwapMouseButton

Syntax
```
function SwapMouseButton(Swap: Bool): Bool;
```

Beschreibung Diese Funktion tauscht die Bedeutung von linker und rechter Maustaste aus. Ist der Parameter *Swap True*, erzeugt die linke Taste Maussignale der rechten Taste und die rechte Taste Maussignale der linken Taste, ist *Swap False*, wird die ursprüngliche Bedeutung der Tasten wiederhergestellt.

Swap: Bestimmt, ob die Bedeutung der Tasten umgekehrt oder die ursprüngliche Bedeutung der Tasten wiederhergestellt werden soll.

Rückgabewert Der Rückgabewert ist *True*, wenn die Funktion die Bedeutung der Maustasten vertauscht hat, andernfalls ist er *False*.

Hinweis Das Umschalten der Tasten ist als Erleichterung für Linkshänder gedacht. Die Funktion *SwapMouseButton* wird gewöhnlich nur von der Systemsteuerung aufgerufen.

Obwohl es Anwendungen freisteht, die Funktion aufzurufen, ist die Maus eine gemeinsam genutzte Ressource, das Vertauschen der Bedeutung der Maustasten beeinflußt also alle Anwendungen.

SwapRecording

Syntax
```
procedure SwapRecording(Flag: Word);
```

Beschreibung Wird mit Microsoft Windows Swap gearbeitet, startet oder beendet diese Prozedur die Analyse des Auslagerungsverhaltens.

Flag: Bestimmt, ob Swap die Analyse starten oder beenden soll. Folgende Werte sind möglich:

Wert	Bedeutung
0	Beendet die Analyse.
1	Auslagerungsaufrufe und -rückgaben aufzeichnen.
2	Wie 1, zusätzlich werden Aufrufe über Thunks aufgezeichnet. Diese Option führt zur Aufzeichnung einer großen Datenmenge.

SwitchStackBack

Syntax
```
procedure SwitchStackBack;
```

Beschreibung Diese Prozedur gibt den Stack der aktuellen Task an das Datensegment der Task zurück, nachdem er vorher durch die Funktion *SwitchStackTo* neuadressiert wurde.

Diese Prozedur hat keine Parameter.

Hinweis Diese Prozedurs ichert beim Rücksprung die Inhalte des AX:DX-Registers.

Querverweis *SwitchStackTo*

SwitchStackTo

Syntax
```
procedure SwitchStackTo(StackSegment, StackPointer,
StackTop: Word);
```

Beschreibung Diese Prozedur ändert den Stack der aktuellen Task auf das durch den Parameter *StackSegment* festgelegte Segment.

Dynamische Linkbibliotheken (DLL) haben keinen Stack; stattdessen verwendet eine DLL den Stack der Task, die die

Bibliothek aufruft. Daher können DLL-Funktionen, die voraussetzen, daß die Inhalte des Code-Segment-Registers (CS) und des Stack-Segment-Registers identisch sind, nicht zum gewünschten Resultat führen. Die Prozedur *SwitchStackTo* adressiert den Stack der Task auf das Datensegment einer DLL um und läßt zu, daß die DLL diese Funktionen aufruft. *SwitchStackTo* kopiert die Argumente der Task auf dem Stack auf die neue Position des Stacks.

StackSegment: Bestimmt das Datensegment, das den Stack enthalten soll.

StackPointer: Legt den Offset des Stack-Anfangs im Datensegment fest.

StackTop: Legt den Offset der Stack-Spitze relativ zum Stack-Anfang fest.

Hinweis

Eine Task kann *SwitchStackTo* aufrufen, bevor eine Funktion in einer DLL aufgerufen wird, die voraussetzt, daß die CS- und DS-Register gleich sind. Beim Rücksprung der DLL-Funktion muß die Task dann *SwitchStackBack* aufrufen, um ihren Stack auf ihr eigenes Datensegment umzuadressieren.

Ebenso kann eine DLL *SwitchStackTo* aufrufen, bevor sie eine Routine aufruft, die die Identität von CS und DS voraussetzt, und kann dann vor der Rückkehr zu der Task, die die DLL-Funktion aufrief, *SwitchStackBack* aufrufen.

Aufrufe von *SwitchStackTo* und *SwitchStackBack* können nicht verschachtelt werden. Das bedeutet, daß nach dem Aufruf von *SwitchStackTo* ein Programm *SwitchStackBack* aufrufen muß, bevor *SwitchStackTo* erneut aufgerufen werden kann.

Querverweis

SwitchStackBack

SyncAllVoices

Syntax

```
function SyncAllVoices: Integer;
```

Beschreibung

Diese Funktion reiht eine Synchronisationsmarkierung in jede Klang-Warteschlange ein.

Nachdem in einer Klang-Warteschlange eine Synchronisationsmarkierung erkannt wurde, wird die Ausgabe abgeschaltet, bis Synchronisationsmarkierungen in allen anderen Warteschlangen aufgefunden werden. Dies bewirkt die Synchronisation aller Tonausgaben.

Diese Funktion hat keine Parameter.

Rückgabewert

Der Rückgabewert ist 0, wenn die Funktion erfolgreich war. Ist eine Klang-Warteschlange voll, dann ist der Rückgabewert *s_SerQFUL*.

TabbedTextOut

Syntax

```
function TabbedTextOut(DC: HDC; X, Y: Integer; Str:
PChar; Count, TabPositions: Integer; var
TabStopPositions; TabOrigin: Integer): LongInt;
```

Beschreibung

Diese Funktion schreibt einen String unter Benutzung der aktuell ausgewählten Schrift in die im Array *TabStopPositions* angegebenen Spalten auf den angegebenen Bildschirm und erweitert dabei Tabulatorzeichen.

DC: Bezeichnet den Gerätekontext.

X: Bestimmt die virtuelle x-Koordinate des Startpunkts des Strings.

Y: Bestimmt die virtuelle y-Koordinate des Startpunkts des Strings.

Str: Zeigt auf den String, der gezeichnet werden soll.

Count: Bestimmt die Anzahl der Zeichen im String.

TabPositions: Bestimmt die Anzahl der Tabulatorpositionen im Array, *TabStopPositions*.

TabStopPositions: Ein Array von Integerzahlen, das die Tabulatorpositionen in Pixeln enthält. Die Tabulatorstops müssen in aufsteigender Reihenfolge sortiert werden; Rückwärtstabulatoren sind nicht zulässig.

TabOrigin: Bestimmt die virtuelle x-Koordinate der Startposition, von der aus die Tabulatoren erweitert werden.

Rückgabewert

Der Rückgabewert bestimmt die Dimensionen des Strings. Die Höhe ist im höherwertigen, die Breite im niederwertigen Word enthalten.

Hinweis

Wenn die Parameter *TabPositions* und *TabStopPositions* 0 sind, werden die Tabulatoren auf acht durchschnittliche Zeichenbreiten erweitert.

Wenn *TabPositions* 1 ist, werden die Tabulatorstops in dem Abstand verteilt, den der erste Wert im Array *TabStopPositions* bestimmt.

Wenn *TabStopPositions* mehr als einen Wert enthält, wird ein Tabulatorstop für jeden Wert im Array gesetzt, bis zu der Zahl, die *TabPositions* angibt.

Der Parameter *TabOrigin* gestattet es einer Anwendung, die Funktion *TabbedTextOut* mehrmals für eine Zeile aufzurufen. Wenn die Anwendung *TabbedTextOut* mehr als einmal aufruft und dabei *TabOrigin* jedesmal auf denselben Wert gesetzt ist, erweitert die Funktion alle Tabulatoren relativ zu der Position, die durch *TabOrigin* vorgegeben ist.

TextOut

Syntax
```
function TextOut(DC: HDC; X, Y: Integer; Str: PChar;
Count: Integer): Bool;
```

Beschreibung Diese Funktion schreibt einen String unter Verwendung der aktuell ausgewählten Schrift auf den angegebenen Bildschirm. Die Startposition des Strings ist durch die Parameter *X* und Y gegeben.

DC: Bezeichnet den Gerätekontext.

X: Bestimmt die virtuelle x-Koordinate des Startpunkts des Strings.

Y: Bestimmt die virtuelle y-Koordinate des Startpunkts des Strings.

Str: Zeigt auf den String, der gezeichnet werden soll.

Count: Bestimmt die Anzahl der Zeichen im String.

Rückgabewert Der Rückgabewert ist *True*, wenn der String gezeichnet wurde, andernfalls ist er *False*.

Hinweis Als Ursprung eines Zeichens ist die obere linke Ecke der Zeichenzelle definiert. Gemäß Voreinstellung wird die aktuelle Position von dieser Funktion nicht benutzt oder aktualisiert.

Eine Anwendung kann aber *SetTextAlign* mit auf *ta_UpdateCP* gesetztem Parameter *wFlags* aufrufen, um es Windows zu ermöglichen, die aktuelle Position jedesmal zu benutzen und zu aktualisieren, wenn die Anwendung *TextOut* für einen gegebenen Gerätekontext aufruft.

Wenn dieses Flag gesetzt ist, ignoriert Windows die Parameter *X* und *Y* nachfolgender Aufrufe von *TextOut*.

Querverweis *SetTextAlign*

Throw

Syntax
```
procedure Throw(var CatchBuf: TCatchBuf; ThrowBack:
Integer);
```

Beschreibung Diese Prozedur stellt die Ausführungsumgebung mit den Werten wieder her, die in *CatchBuf* gespeichert sind.

Die Ausführungsumgebung ist der Status aller Systemregister und des Befehlszählers. Die Ausführung fährt mit der Funktion *Catch* fort, die die Umgebung kopiert hat.

Der Parameter *ThrowBack* wird als Rückgabewert an die Funktion *Catch* übergeben, dieser kann ungleich 0 sein.

CatchBuf: Ein Array, das die Ausführungsumgebung enthält.

ThrowBack: Bestimmt den Wert, der an die Funktion *Catch* übergeben werden soll.

ToAscii

Syntax

```
function ToAscii(VirtKey, ScanCode: Word; KeyState:
PChar; CharBuff: Pointer; Flags: Word): Integer;
```

Beschreibung

Diese Funktion übersetzt den durch den Parameter *VirtKey* bestimmten virtuellen Tastencode und den durch den Parameter *KeyState* bestimmten aktuellen Tastaturstatus in das oder die zugehörige(n) ANSI-Zeichen.

VirtKey: Bestimmt den virtuellen Tastencode, der übersetzt werden soll.

ScanCode: Bestimmt den ursprünglichen Hardware-Scan-Code der Taste, die übersetzt werden soll. Das höchstwertige Bit dieses Wertes wird gesetzt, wenn die Taste nicht gedrückt ist.

KeyState: Zeigt auf ein Array von 256 Bytes, von denen jedes den Status einer Taste enthält. Wenn das höchstwertige Bit des Bytes gesetzt ist, ist die Taste nicht gedrückt.

CharBuff: Zeigt auf einen 32-Bit-Puffer, der das oder die übersetzten ANSI-Zeichen aufnimmt.

Flags: Das Bit 0 wirkt als Flag für die Menüanzeige.

Rückgabewert

Der Rückgabewert gibt die Zahl der Zeichen an, die in den Puffer kopiert wurden, auf den der Parameter *CharBuff* zeigt. Der Rückgabewert ist negativ, wenn die Taste eine tote Taste war, andernfalls ist er einer der folgenden Werte:

Wert	Bedeutung
2	Es wurden zwei Zeichen in den Puffer kopiert. Das sind gewöhnlich ein Akzent und das Zeichen einer tote Taste, falls die tote Taste nicht anders übersetzt werden kann.
1	Es wurde ein ANSI-Zeichen in den Puffer kopiert.
0	Die angegebene virtuelle Taste hat für den aktuellen Tastaturstatus keine Übersetzung.

Hinweis

Die Argumente, die an die Funktion *ToAscii* übergeben werden, reichen eventuell nicht aus, den virtuellen Tastencode zu übersetzen, weil vorher eine tote Taste im Tastaturtreiber gespeichert wurde. Üblicherweise führt *ToAscii* die Übersetzung gemäß des virtuellen Tastencodes durch. In einigen Fällen wird jedoch der Parameter *ScanCode* verwendet, um zwischen dem Drücken und dem Loslassen einer Taste zu unterscheiden. Der Scan-Code wird für die Übersetzung der Kombinationen von Alt mit numerischen Tasten benutzt.

TrackPopupMenu

Syntax
```
function TrackPopupMenu(Menu: HMenu; Flags: Word; x, y,
Reserved: Integer; Wnd: HWnd; Reserved: Pointer): Bool;
```

Beschreibung Diese Funktion stellt ein Pop-Up-Menü an der angegebenen Position dar und ermittelt die Tracking-Auswahl der Optionen aus dem Pop-Up-Menü.

Ein Pop-Up-Menü für Tracking-Auswahl kann irgendwo auf dem Bildschirm erscheinen. Der Parameter *Menu* gibt das Handle des darzustellenden Menüs an. Die Funktion ruft dieses Handle durch Aufruf von *CreatePopupMenu* ab, um ein neues Pop-Up-Menü anzulegen oder durch Aufruf von *GetSubMenu*, um das Handle eines Pop-Up-Menüs, das einer vorhandenen Menüoption zugeordnet ist, wiederzuerhalten. Windows sendet Botschaften, die vom Menü erzeugt wurden, an das durch den Parameter *Wnd* bezeichnete Fenster.

Menu: Bezeichnet das Pop-Up-Menü, das dargestellt werden soll.

Flags: Wird nicht verwendet. Dieser Parameter muß auf 0 gesetzt werden.

x: Bestimmt die horizontale Position der linken Seite des Menüs auf dem Bildschirm in Bildschirmkoordinaten.

y: Bestimmt die vertikale Position der Oberkante des Menüs auf dem Bildschirm in Bildschirmkoordinaten.

Reserved: Ist reserviert und muß auf 0 gesetzt werden.

Wnd: Bezeichnet das Fenster, das das Pop-Up-Menu besitzt. Dieses Fenster empfängt alle *wm_Command*- Botschaften vom Menü.

Reserved: Ist reserviert und muß auf **nil** gesetzt werden.

Rückgabewert Der Rückgabewert gibt das Ergebnis der Funktion an. Er ist *True*, wenn die Funktion erfolgreich war, andernfalls ist er *False*.

Querverweis *CreatePopupMenu, GetSubMenu*

TranslateAccelerator

Syntax
```
function TranslateAccelerator(Wnd: HWnd; AccTable:
THandle; var Msg: TMsg): Integer;
```

Beschreibung Diese Funktion bearbeitet Accelerator-Tasten für Menübefehle. *TranslateAccelerator* übersetzt die *wm_KeyUp*- und *wm_KeyDown*-Botschaften in *wm_Command*- oder *wm_SysCommand*-Botschaften, wenn in der Accelerator-Tabelle der Anwendung

ein Eintrag für die Taste vorhanden ist. Das höherwertige Word des Parameters *lParam* der Botschaft *wm_Command* oder *wm_SysCommand* enthält den Wert 1, um diese Botschaft von anderen Botschaften, die von Menüs oder Steuerelementen gesendet wurden, zu unterscheiden. Die Botschaften *wm_Command* oder *wm_SysCommand* werden direkt an das Fenster gesendet und nicht in die Anwendungs-Warteschlange gestellt. Die Funktion *TranslateAccelerator* kehrt nicht zurück, bis die Botschaft bearbeitet ist. Accelerator-Tasten, die dazu definiert sind, Optionen aus dem Systemmenü auszuwählen, werden in *wm_SysCommand*-Botschaften übersetzt, alle anderen werden in *wm_Command*-Botschaften übersetzt.

Wnd: Bezeichnet das Fenster, dessen Botschaften übersetzt werden sollen.

AccTable: Bezeichnet eine Accelerator-Tabelle, die mit der Funktion *LoadAccelerators* geladen wurde.

Msg: Eine *TMsg*-Struktur mit Botschaftsinformationen, die mit *PeekMessage* oder *GetMessage* abgerufen werden.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn die Übersetzung stattfindet, andernfalls ist er 0.

Hinweis

Wenn *TranslateAccelerator* einen Wert ungleich 0 zurückgibt (die Botschaft also übersetzt ist), sollte die Anwendung die gleiche Botschaft nicht nochmals mit der Funktion *TranslateMessage* bearbeiten. Befehle in Akccelerator-Tabellen müssen keine Entsprechungen in den Menüoptionen haben.

Entspricht der Accelerator-Befehl der Menüoption, werden der Anwendung *wm_InitMenu-* und *wm_InitMenuPopup*-Botschaften übergeben, so als ob der Benutzer versuchen würde, das Menü anzuzeigen. Diese Botschaften werden jedoch nicht gesendet, wenn eine der folgenden Bedingungen erfüllt ist:

- Das Fenster ist gesperrt.
- Die Menüoption ist gesperrt.
- Der Befehl ist nicht im Systemmenü und das Fenster ist verkleinert.
- Ein Mausfang ist aktiv (siehe auch: Funktion *SetCapture*).

Wenn das Fenster das aktive Fenster ist und es keinen Tastaturfokus gibt (das trifft im allgemeinen zu, wenn das Fenster verkleinert ist), werden *wm_SysKeyUp-* und *wm_SysKeyDown*-Botschaften anstelle von *wm_KeyUp-* und *wm_KeyDown*-Botschaften gesendet. Wenn eine einer Menüoption entsprechende Accelerator-Eingabe auftritt, während das Fenster, welches das Menü besitzt, als Symbol dargestellt ist, wird keine *wm_Command*-Botschaft gesendet. Wenn jedoch eine Accele-

rator-Eingabe auftritt, die mit keiner der Optionen im Menü des Fensters übereinstimmt, so wird eine *wm_Command*-Botschaft gesendet – auch dann, wenn das Fenster als Symbol dargestellt ist.

TranslateMDISysAccel

Syntax
```
function TranslateMDISysAccel(Wnd: HWnd; var Msg: TMsg): Bool;
```

Beschreibung Diese Funktion bearbeitet Accelerator-Tasten für Systemmenübefehle untergeordneter Fenster von Multidokumentschnittstellen (MDI).

TranslateMDISysAccel übersetzt *wm_KeyUp*- und *wm_KeyDown*-Botschaften in *wm_SysCommand*-Botschaften. Dabei hat das höherwertige Word des Parameters *lParam* der *wm_SysCommand*-Botschaft den Wert 1, um diese Botschaft von Botschaften zu unterscheiden, die von Menüs oder Steuerelementen gesendet werden.

WndClient: Bezeichnet das übergeordnete MDI-Client-Fenster.

Msg: Eine *TMsg*-Struktur mit Botschaftsinformationen, die mit *PeekMessage* oder *GetMessage* abgerufen werden.

Rückgabewert Der Rückgabewert ist *True*, wenn die Funktion eine Botschaft in einen Systembefehl übersetzt hat, andernfalls ist er *False*.

TranslateMessage

Syntax
```
function TranslateMessage(var Msg: TMsg): Bool;
```

Beschreibung Diese Funktion übersetzt virtuelle Tastenbotschaften in Zeichenbotschaften wie folgt:

wm_KeyDown/wm_KeyUp-Kombinationen produzieren eine Botschaft *wm_Char* oder *wm_DeadChar*.

wm_SysKeyDown/wm_SysKeyUp-Kombinationen produzieren eine Botschaft *wm_SysChar* oder *wm_SysDeadChar*.

Die Zeichenbotschaften werden in die Anwendungs-Warteschlange eingereiht, um beim nächsten Aufruf der Funktion *GetMessage* oder *PeekMessage* durch die Anwendung gelesen zu werden.

Msg: Eine *TMsg*-Struktur mit Botschaftsinformationen, die mit *PeekMessage* oder *GetMessage* abgerufen werden.

Rückgabewert Der Rückgabewert ist *True*, wenn die Botschaft übersetzt wurde (d.h. Zeichenbotschaften in die Anwendungs-Warteschlange gestellt werden), andernfalls ist er *False*.

Hinweis *TranslateMessage* verändert die durch den Parameter *Msg* bezeichnete Botschaft nicht. *TranslateMessage* erzeugt *wm_Char*-Botschaften nur für die Tasten, denen durch den Tastaturtreiber ASCII-Zeichen zugeordnet sind. Eine Anwendung sollte *TranslateMessage* nicht aufrufen, wenn die Anwendung virtuelle Tastenbotschaften für irgendwelche anderen Zwecke verarbeitet. Zum Beipiel sollte eine Anwendung *TranslateMessage* nicht aufrufen, wenn die Funktion *TranslateAccelerator* einen Wert ungleich 0 zurückgibt.

TransmitCommChar

Syntax
```
function TransmitCommChar(Cid: Integer; Chr: Char):
Integer;
```

Beschreibung Diese Funktion markiert das durch den Parameter *Chr* angegebene Zeichen zur sofortigen Übertragung, indem sie es an den Anfang der Übertragungs-Warteschlange stellt.

Cid: Bestimmt die Schnittstelle, die das Zeichen empfangen soll. Die Funktion *OpenComm* gibt diesen Wert zurück.

Chr: Bestimmt das zu übertragende Zeichen.

Rückgabwert Der Rückgabewert ist 0, wenn die Funktion erfolgreich war und negativ, wenn das Zeichen nicht übertragen werden kann. Ein Zeichen kann nicht übertragen werden, wenn das durch den vorherigen Aufruf der Funktion *TransmitCommChar* bestimmte Zeichen noch nicht gesendet wurde.

Querverweis *OpenComm*

UngetCommChar

Syntax
```
function UngetCommChar(Cid: Integer; Chr: Char):
Integer;
```

Beschreibung Diese Funktion stellt das durch den Parameter *Chr* bestimmte Zeichen in die Empfangs-Warteschlange zurück und macht es zu dem Zeichen, das bei einem darauffolgenden Lesevorgang als erstes aus der Warteschlange gelesen wird.

Nachfolgende Aufrufe der Funktion *UngetCommChar* sind nicht erlaubt. Das Zeichen, das in die Warteschlange zurückgestellt wurde, muß vor dem Versuch, ein weiteres Zeichen zurückzustellen, gelesen werden.

Cid: Bestimmt das Datenübertragungsgerät, welches das Zeichen empfangen soll.

Chr: Bestimmt das Zeichen, das in die Empfangs-Warteschlange zurückgestellt werden soll.

Rückgabewert Der Rückgabewert ist 0, wenn die Funktion erfolgreich war und negativ, wenn ein Fehler auftritt.

Querverweis *OpenComm*

UnhookWindowsHook

Syntax
```
function UnhookWindowsHook(Hook: Integer; HookFunc:
TFarProc): Bool;
```

Beschreibung Diese Funktion entfernt die Windows-Filterverwaltungsfunktion, die der Parameter *Hook* bestimmt, aus einer Kette von Filterverwaltungsfunktionen.

Eine Windows-Filterverwaltungsfunktion verarbeitet Ereignisse, bevor diese an die Botschaftsschleife einer Anwendung in der Funktion *WinMain* gesendet wird.

Hook: Bestimmt den Typ der zu entfernenden Filterverwaltungsfunktion. Er kann einen der folgenden Werte annehmen.

Wert	Bedeutung
wh_CallWndProc	Installiert einen Fensterfunktionsfilter.
wh_GetMessage	Installiert einen Botschaftsfilter.
wh_JournalPlayback	Installiert einen Filter für ein Wiedergabeprotokoll.
wh_JournalRecord	Installiert einen Filter für ein Aufzeichnungsprotokoll.
wh_Keyboard	Installiert einen Tastaturfilter.
wh_MsgFilter	Installiert einen Botschaftsfilter.

HookFunc: Ist die Adresse der Prozedurinstanz der Filterverwaltungsfunktion.

Rückgabewert Der Rückgabewert ist *True*, wenn die Filterverwaltungsfunktion erfolgreich entfernt wurde, andernfalls ist er *False*.

UnionRect

Syntax
```
function UnionRect(var DestRect, Src1Rect, Src2Rect:
TRect): Integer;
```

Beschreibung Diese Funktion erzeugt die Vereinigung zweier Rechtecke. Die Vereinigung ist das kleinste Rechteck, das beide Quellrechtecke einschließt.

DestRect: Eine *TRect*-Datenstruktur, die das neue Rechteck aufnehmen soll.

Src1Rect: Eine *TRect*-Datenstruktur, die ein Quellrechteck enthält.

Src2Rect: Eine *TRect*-Datenstruktur, die ein Quellrechteck enthält.

Rückgabewert Der Rückgabewert ist ungleich 0, wenn die Vereinigung nicht leer ist und 0, wenn die Vereinigung leer ist.

Hinweis Windows ignoriert die Abmessungen eines leeren Rechtecks (also eines Rechtecks, das keine Höhe oder keine Breite hat).

UnlockData

Syntax
```
function UnlockData(Dummy: Integer): THandle;
```

Beschreibung Diese Inline-Funktion hebt die Sperre des aktuellen Datensegments auf. Sie ist für die Verwendung von Modulen vorgesehen, die verschiebbare Datensegmente haben.

Dummy: Wird nicht verwendet und kann auf 0 gesetzt werden.

UnlockResource

Syntax
```
function UnlockResource(ResData: THandle): Bool;
```

Beschreibung Diese Inline-Funktion hebt die Sperre der durch den Parameter *ResData* angegebene Ressource auf und verringert den Referenzzähler der Ressource um 1.

ResData: Bezeichnet den globalen Speicherblock, dessen Sperre aufgehoben werden soll.

Rückgabewert Der Rückgabewert ist 0, wenn der Referenzzähler des Blocks bis auf 0 heruntergesetzt wurde, andernfalls ist er ungleich 0.

UnlockSegment

Syntax
```
function UnlockSegment(Segment: Word): THandle;
```

Beschreibung Diese Funktion hebt die Sperre des Segments auf, dessen Segmentadresse durch den Parameter *Segment* angegeben ist. Wenn *Segment* -1 ($FFFF) ist, hebt die Funktion *UnlockSegment* die Sperre des aktuellen Datensegments auf.

Im Real Mode oder wenn das Segment verwerfbar ist, verringert *UnlockSegment* den Sperrenzähler des Segments um 1. Im Protected Mode verringert *UnlockSegment* nur den Sperrenzähler verwerfbarer Objekte und automatischer Datensegmente. Wenn der Sperrenzähler auf 0 zurückgesetzt ist, ist die Sperre des Segments vollständig aufgehoben und das Segment kann verschoben oder verworfen werden.

Andere Funktionen können den Sperrenzähler eines Speicherobjektes ebenfalls beeinflussen. In der Beschreibung der Funktion *GlobalFlags* finden Sie eine Liste dieser Funktionen.

Eine Anwendung muß jeden Aufruf von *LockSegment* für ein Segment mit einem entsprechenden *UnlockSegment* wieder aufheben.

Segment: Bestimmt die Segmentadresse des Segments, dessen Sperre aufgehoben werden soll. Wenn *Segment* -1 ist, hebt *UnlockSegment* die Sperre des aktuellen Datensegments auf.

Rückgabewert Der Rückgabewert ist 0, wenn der Sperrenzähler des Segments auf 0 heruntergesetzt wurde, andernfalls ungleich 0.

Eine Anwendung sollte sich nicht auf den Rückgabewert verlassen, um zu bestimmen, wie oft nacheinander sie *UnlockSegment* für das Segment aufrufen muß.

Querverweis *LockSegment*

UnrealizeObject

Syntax
```
function UnrealizeObject(hObject: HBrush): Bool;
```

Beschreibung Bezeichnet der Parameter *hObject* einen Pinsel, weist diese Funktion die GDI an, den Ursprung dieses Pinsels zurückzusetzen, wenn er das nächste Mal ausgewählt wird. Bezeichnet *hObject* eine virtuelle Palette, weist diese Funktion die GDI an, eine Palette zu realisieren, als wäre sie vorher noch nicht realisiert worden. Ruft die Anwendung das nächste Mal die Funktion *RealizePalette* für die angegebene Palette auf, legt die GDI die virtuelle Palette in der Systempalette komplett neu an.

hObject: Bezeichnet das Objekt, das zurückgesetzt werden soll.

Rückgabewert Der Rückgabewert ist *True*, wenn die Funktion erfolgreich war, andernfalls ist er *False*.

Hinweis Die Funktion *UnrealizeObject* sollte nicht für vordefinierte Objekte verwendet werden. Diese Funktion muß immer aufgerufen werden, wenn ein neuer Pinselursprung mit der Funktion *SetBrushOrg* gesetzt wird. Ein Pinsel, der durch den Parameter *hObject* bezeichnet wird, darf nicht der aktuell ausgewählte Pinsel eines Bildschirmkontextes sein. Eine Palette, die durch *hObject* angegeben ist, kann die aktuell ausgewählte Palette eines Bildschirmkontextes sein.

UnregisterClass

Syntax
```
function UnregisterClass(ClassName: PChar; Instance:
THandle): Bool;
```

Beschreibung Diese Funktion löscht die die durch *ClassName* bestimmte Fensterklasse aus der Fensterklassentabelle und gibt dabei den für die Klasse erforderlichen Speicher frei.

ClassName: Zeigt auf einen nullterminierten String, der die Klassenbezeichnung enthält. Diese Klassenbezeichnung muß vorher durch Aufruf der Funktion *RegisterClass* mit einem gültigen Feld *hInstance* in der *TWndClass*-Struktur registriert werden. Vordefinierte Klassen wie beispielsweise die von Dialogfenstersteuerelementen können nicht gelöscht werden.

lInstance: Bezeichnet die Instanz des Moduls, das die Klasse erzeugt hat.

Rückgabewert Der Rückgabewert ist *True*, wenn die Funktion die Fensterklasse erfolgreich aus der Fensterklassentabelle gelöscht hat. Er ist *False*, wenn die Klasse nicht gefunden werden konnte oder wenn ein Fenster existiert, das mit dieser Klasse erzeugt wurde.

Hinweis Bevor diese Funktion benutzt wird, müssen alle Fenster, die mit der angegebenen Klasse erzeugt wurden, entfernt und freigegeben werden.

Querverweis *RegisterClass*

UpdateColors

Syntax
```
function UpdateColors(DC: HDC): Integer;
```

Beschreibung Diese Funktion aktualisiert den Client-Bereich des durch den Parameter *DC* bezeichneten Gerätekontextes, indem die aktuellen Farben im Client-Bereich pixelweise an die Systempalette angeglichen werden. Ein inaktives Fenster mit einer realisierten virtuellen Palette kann *UpdateColors* als Alternative zum Neuzeichnen seines Client-Bereiches aufrufen, wenn sich die Systempalette ändert.

DC: Bezeichnet den Gerätekontext.

Rückgabewert Der Rückgabewert wird nicht verwendet.

Hinweis *UpdateColors* aktualisiert einen Client-Bereich normalerweise schneller, als er neu gezeichnet werden kann. Da *UpdateColors* jedoch die Farbumwandlung gemäß der Farbe eines jeden Pixels vor der Änderung der Systempalette ausführt, bewirkt jeder Aufruf dieser Funktion einen gewissen Verlust an Farbwiedergabetreue.

UpdateWindow

Syntax
```
procedure UpdateWindow(Wnd: HWnd);
```

Beschreibung Diese Prozedur aktualisiert den Client-Bereich des gegebenen Fensters, indem sie eine *wm_Paint*-Botschaft an das Fenster sendet, wenn der Aktualisierungsbereich für das Fenster nicht leer ist. Die Prozedur *UpdateWindow* sendet eine *wm_Paint-*

Botschaft unter Umgehung der Anwendungs-Warteschlange direkt an die Fensterfunktion des gegebenen Fensters. Ist die Aktualisierungsregion leer, wird keine Botschaft gesendet.

Wnd: Bezeichnet das zu aktualisierende Fenster.

ValidateCodeSegments

Syntax
```
procedure ValidateCodeSegments;
```

Beschreibung Diese Prozedur gibt Debugging-Informationen auf ein Terminal aus, wenn Codesegmente durch zufällige Speicherüberschreibungen verändert wurden. Die Prozedur ist nur in der Debugging-Version von Windows verfügbar und standardmäßig aktiviert. Um sie zu deaktivieren, muß das Flag *EnableSegmentChecksum* im Abschnitt [kernel] der Datei WIN.INI auf 0 gesetzt werden. Im Protected Mode (Standard- oder erweiterter 386-Modus) überprüft Windows die Gültigkeit der Codesegmente nicht.

Diese Prozedur hat keine Parameter.

ValidateFreeSpaces

Syntax
```
function ValidateFreeSpaces: Pointer;
```

Beschreibung Diese Funktion (nur in der Debugging-Version von Windows verfügbar) überprüft freie Segmente auf gültige Inhalte. In der Debugging-Version von Windows füllt das Kernel alle Bytes in freien Segmenten mit dem Hexadezimalwert $CC.

Diese Funktion beginnt mit der Überprüfung im freien Segment mit der niedrigsten Adresse und fährt solange fort, bis ein ungültiges Byte gefunden wird oder die Überprüfung ergibt, daß der gesamte freie Speicher gültige Inhalte hat. Vor Aufruf dieser Funktion müssen Sie die Datei WIN.INI um folgende Zeilen ergänzen:

```
[kernel]
EnableFreeChecking=1
EnableHeapChecking=1
```

Diese Funktion hat keine Parameter.

Hinweis Windows übergibt die Debugging-Informationen an das Debugging-Terminal, wenn ein ungültiges Byte gefunden wird. Anschließend wird ein *Fatal Exit* ausgeführt. Die [kernel]-Einträge in WIN.INI sorgen für die automatische Überprüfung des freien Speichers.

Bevor ein Speicherblock als Reaktion auf einen Aufruf der Funktion *GlobalAlloc* zurückgegeben wird, überprüft Windows diesen Speicher, um sicherzustellen, daß er mit $CC gefüllt ist.

Vor Aufruf der Funktion *GlobalCompact* wird der gesamte freie Speicher überprüft.

Der Aufruf dieser Funktion verlangsamt Windows um etwa 20%.

ValidateRect

Syntax
```
procedure ValidateRect(Wnd: HWnd; Rect: PRect);
```

Beschreibung Diese Prozedur führt eine Gültigkeitsüberprüfung des Client-Bereichs innerhalb des gegebenen Rechtecks durch, indem sie das Rechteck aus der Aktualisierungsregion des angegebenen Fensters löscht.

Wenn der Parameter *Rect* 0 ist, wird das gesamte Fenster auf Gültigkeit überprüft.

Wnd: Bezeichnet das Fenster, dessen Aktualisierungsregion verändert werden soll.

Rect: Zeigt auf eine *TRect*-Datenstruktur, die das Rechteck, das aus der Aktualisierungsregion gelöscht werden soll, in Client-Koordinaten enthält.

Hinweis Die Funktion *BeginPaint* führt automatisch eine Gültigkeitsprüfung für den gesamten Client-Bereich durch. Wenn nur ein Teil der Aktualisierungsregion auf Gültigkeit überprüft werden muß, sollte keine der Prozeduren *ValidateRect* und *ValidateRgn* aufgerufen werden, bevor die nächste Botschaft *wm_Paint* erzeugt wird.

Windows erzeugt solange *wm_Paint*-Botschaften, bis die Gültigkeitsüberprüfung der gegenwärtigen Aktualisierungsregion durchgeführt ist.

Querverweis *BeginPaint*

ValidateRgn

Syntax
```
procedure ValidateRgn(Wnd: HWnd; Rgn: HRgn);
```

Beschreibung Diese Prozedur führt eine Gültigkeitsüberprüfung des Client-Bereichs innerhalb der gegebenen Region durch, indem sie die Region aus der Aktualisierungsregion des angegebenen Fensters löscht.

Ist der Parameter *Rgn* 0, wird das gesamte Fenster auf Gültigkeit überprüft.

Wnd: Bezeichnet das Fenster, dessen Aktualisierungsregion verändert werden soll.

Rgn: Bezeichnet eine Region, die einen Bereich definiert, der aus der Aktualisierungsregion gelöscht werden soll.

Hinweis Die angegebene Region muß vorher mit einer Region-Funktion erstellt worden sein. Die Koordinaten der Region müssen als Client-Koordinaten vorliegen.

VkKeyScan

Syntax
```
function VkKeyScan (Chr: Word): Word;
```

Beschreibung Diese Funktion übersetzt ein ANSI-Zeichen in den zugehörigen virtuellen Tastencode und Shift-Status für die aktuelle Tastatur.

VkKeyScan wird von Anwendungen benutzt, die Zeichen mit den Botschaften *wm_KeyUp* und *wm_KeyDown* senden.

Chr: Bestimmt das Zeichen, für das der zugehörige virtuelle Tastencode gefunden werden soll.

Rückgabewert Der virtuelle Tastencode wird im niederwertigen Byte zurückgegeben, der Shift-Status im höherwertigen Byte. Mögliche Werte sind:

Wert	Bedeutung
0	Kein Shift.
1	Zeichentaste ist zusammen mit der Shift -Taste gedrückt.
2	Zeichen ist ein Steuerzeichen.
6	Zeichen ist Ctrl + Alt .
7	Zeichen ist Shift + Ctrl + Alt .
3, 4, 5	Eine Shift-Tastenkombination, die nicht für Zeichen benutzt wird.

Wird keine Taste gefunden, deren Übersetzung dem übergebenen ANSI-Code entspricht, wird sowohl im höherwertigen als auch im niederwertigen Byte -1 zurückgegeben.

Hinweis Übersetzungen für das numerische Tastenfeld (*vk_NumPad0* bis *vk_Divide*) werden ignoriert. Diese Funktion ist dafür vorgesehen, nur eine Übersetzung für die Haupttastatur vorzunehmen.

WaitMessage

Syntax
```
procedure WaitMessage;
```

Beschreibung Diese Prozedur gibt die Programmsteuerung an andere Anwendungen weiter, wenn eine Anwendung keine anderen Aufgaben auszuführen hat.

WaitMessage unterbricht die Anwendung und kehrt solange nicht zurück, bis eine neue Botschaft in die Anwendungs-Warteschlange gestellt wird. Diese Prozedur hat keine Parameter.

Hinweis

GetMessage, *PeekMessage* und *WaitMessage* übergeben die Programmsteuerung an andere Anwendungen. Aufrufe dieser Funktionen sind die einzige Möglichkeit, andere Anwendungen ablaufen zu lassen.

Wenn *GetMessage*, *PeekMessage* und *WaitMessage* die Programmsteuerung an andere Anwendungen abgeben, können sich der Stack und die Datensegmente der die Funktion aufrufenden Anwendung im Arbeitsspeicher verschieben, um sich veränderten Speicheranforderungen anderer Anwendungen anzupassen. Hält die Anwendung Zeiger auf Objekte im Daten- oder Stack-Bereich gespeichert (d.h. globale oder lokale Variablen), können diese Zeiger nach einem Aufruf von *GetMessage*, *PeekMessage* oder *WaitMessage* ungültig werden.

WaitSoundState

Syntax

```
function WaitSoundState(State: Integer): Integer;
```

Beschreibung

Diese Funktion wartet, bis der Klangtreiber den angegebenen Zustand erreicht hat.

State: Bestimmt den Zustand der Klang-Warteschlangen. Er kann einen der folgenden Werte annehmen:

Wert	Bedeutung
s_AllThreshlod	Alle Klang-Warteschlangen haben den Schwellenwert erreicht.
s_QueueEmprty	Alle Klang-Warteschlangen sind leer und die Klangtreiber ausgeschaltet.
s_Threshold	Eine Klang-Warteschlange hat den Schwellenwert erreicht und gibt den Klang aus.

Rückgabewert

Der Rückgabewert ist 0, wenn die Funktion erfolgreich war, wenn der Zustand ungültig ist, ist er *s_SerDST*.

WindowFromPoint

Syntax

```
function WindowFromPoint(Point: TPoint): HWnd;
```

Beschreibung

Diese Funktion ermittelt das Fenster, das den vorgegebenen Punkt enthält. Der Parameter *Point* muß die Bildschirmkoordinaten eines Punktes auf dem Bildschirm bestimmen.

Point: Eine *TPoint*-Datenstruktur, die den zu untersuchenden Punkt definiert.

Rückgabewert

Der Rückgabewert bezeichnet das Fenster, in dem der Punkt liegt. Er ist 0, wenn an dem angegebenen Punkt kein Fenster existiert.

WinExec

Syntax

```
function WinExec(CmdLine: PChar; CmdShow: Word): Word;
```

Beschreibung

Diese Funktion bringt die durch den Parameter *CmdLine* bezeichnete Windows- oder Nicht-Windows-Anwendung zur Ausführung.

Der Parameter *CmdShow* bestimmt den Anfangszustand des Hauptfensters der Anwendung, wenn es erstellt wird.

CmdLine: Zeigt auf einen nullterminierten String, der die Befehlszeile für die auszuführende Anwendung enthält (Dateiname und optionale Parameter). Enthält der String *CmdLine* keinen Verzeichnispfad, sucht Windows nach der ausführbaren Datei in folgender Reihenfolge:

1. Im aktuellen Verzeichnis

2. Im Windows-Verzeichnis (dem Verzeichnis, das WIN.COM enthält); die Funktion *GetWindowsDirectory* ermittelt den Pfadnamen dieses Verzeichnisses.

3. Im Windows-Systemverzeichnis (dem Verzeichnis, das Systemdateien wie KERNEL.EXE enthält); die Funktion *GetSystemDirectory* ermittelt den Pfadnamen des Verzeichnisses.

4. In den Verzeichnissen, die in der Environment-Variablen PATH aufgeführt sind.

5. In der Liste von Verzeichnissen, die in einem Netz mit dem Befehl *MAP* angegeben wurden.

Wenn der Dateiname einer Anwendung keine andere Namenserweiterung enthält, wird .EXE angenommen.

CmdShow: Beschreibt, wie ein Windows-Anwendungsfenster dargestellt werden soll. Die gültigen Werte für den Parameter finden Sie in Kapitel 1 »Windows-Stile und -Konstanten« im Abschnitt »sw_Xxxx«. Bei einer Nicht-Windows-Anwendung legt die .PIF-Datei den Fensterstatus fest.

Rückgabewert

Wenn die Funktion erfolgreich war, ist der Rückgabewert größer als 32, andernfalls ist er ein Wert kleiner als 32, der den Fehler näher bestimmt.

Die folgende Liste beschreibt die Fehlerwerte, die von dieser Funktion zurückgegeben werden:

Wert	Bedeutung
0	Zu wenig freier Speicher.
2	Datei nicht gefunden.
3	Pfad nicht gefunden.
5	Versuch, eine Task dynamisch einzubinden.

Wert	Bedeutung
6	Bibliothek erfordert getrennte Datensegmente für jede Task.
10	Falsche Windows-Version.
11	Ungültige .EXE-Datei (Keine Windows-.EXE-Datei oder Fehler im .EXE-Darstellungsformat).
12	OS/2-Anwendung.
13	DOS 4.0-Anwendung.
14	Unbekannter .EXE-Typ.
15	Versuch, im Protected Mode (Standardmodus oder erweiterter 386-Modus) eine für frühere Windows-Versionen erstellte .EXE-Datei zu laden.
16	Versuch, eine zweite Instanz einer .EXE-Datei zu laden, die mehrere beschreibbare Datensegmente enthält.
17	Versuch, im Large-Frame-EMS-Modus eine zweite Instanz einer Anwendung zu laden, die mit bestimmten nicht gemeinsam benutzbaren DLLs verbunden ist, die bereits in Benutzung sind.
18	Versuch, eine Anwendung im Real Mode zu laden, die für Protected Mode gekennzeichnet ist.

Hinweis Die Funktion *LoadModule* bietet eine alternative Methode, ein Programm auszuführen.

WinHelp

Syntax
```
function WinHelp(Wnd: HWnd; HelpFile: PChar; Command:
Word; Data: LongInt): Bool;
```

Beschreibung Diese Funktion ruft die Windows-Hilfeanwendung auf und übergibt optionale Daten, die die Art der von der Anwendung angeforderten Hilfe festlegen. Die Anforderung gibt den Namen und, falls nötig, den Verzeichnispfad der Hilfedatei an, die die Hilfeanwendung anzeigen soll.

Wnd: Bezeichnet das Fenster, das die Hilfe anfordert.

HelpFile: Zeigt auf einen nullterminierten String, der den Namen der Hilfedatei und, falls nötig, den Verzeichnispfad enthält, die von der Hilfeanwendung angezeigt werden soll.

Command: Bestimmt die Art der angeforderten Hilfe. Er kann einen der folgenden Werte annehmen:

Wert	Bedeutung
Help_Context	Zeigt Hilfe für einen bestimmten Kontext an, der durch den LongInt-Parameter *Data* bezeichnet wird.

Wert	Bedeutung
Help_HelpOnHelp	Zeigt Hilfe für die Benutzung der Hilfeanwendung selbst an. Wenn der Parameter *Command* auf *Help_HelpOnHelp* gesetzt wird, ignoriert *WinHelp* die Parameter *HelpFile* und *Data*.
Help_Index	Zeigt den Index der angegebenen Hilfedatei an. Eine Anwendung sollte diesen Wert nur für Hilfedateien mit einem einzigen Index benutzen und sie sollte diesen Wert nicht mit *Help_SetIndex* benutzen.
Help_Key	Zeigt Hilfe für ein einzelnes Schlüsselwort an, das durch einen String bezeichnet wird, auf den *Data* zeigt.
Help_MultiKey	Zeigt Hilfe für ein Schlüsselwort in einer veränderbaren Schlüsselworttabelle an.
Help_Quit	Teilt der Hilfeanwendung mit, daß die angegebene Hilfedatei nicht mehr in Gebrauch ist.
Help_SetIndex	Setzt den durch *Data* bestimmten Kontext als aktuellen Index für die durch *HelpFile* angegebene Hilfedatei. Dieser Index bleibt solange aktuell, bis der Benutzer auf eine andere Hilfedatei zugreift. Um sicherzustellen, daß der richtige Index gesetzt bleibt, sollte die Anwendung nach jedem *WinHelp*-Aufruf mit auf *Help_Context* gesetztem Parameter *Command* die Funktion *WinHelp* erneut aufrufen, wobei *Command* auf *Help_SetIndex* gesetzt sein sollte (und *Data* den zugehörigen Kontextbezeichner angibt). Eine Anwendung sollte diesen Wert nur für Hilfedateien mit mehr als einem Index benutzen und sie sollte diesen Wert nicht mit *Help_Index* verwenden.

Data: Bestimmt den Kontext oder das Schlüsselwort für die angeforderte Hilfe.

- Hat *Command* den Wert *Help_Context*, ist *Data* ein LongInt, vorzeichenloser 32-Bit-Integer der eine Zahl für den Kontextbezeichner enthält.

- Hat *Command* den Wert *Help_Key*, ist *Data* ein Zeiger auf einen mit 0 abgeschlossenen String, der ein Schlüsselwort für die Thematik der Hilfe enthält.

- Hat *Command* den Wert *Help_MultiKey*, ist *Data* ein Zeiger auf eine *TMultiKeyHelp*-Datenstruktur.

- In allen anderen Fällen wird *Data* ignoriert und sollte auf 0 gesetzt werden.

Rückgabewert Der Rückgabewert ist *True*, wenn die Funktion erfolgreich war, andernfalls ist er *False*.

Hinweis Die Anwendung muß *WinHelp* mit auf *Help_Quit* gesetztem *Command* aufrufen, bevor sie das Fenster, das die Hilfe anforderte, schließt.

Die Hilfeanwendung wird solange nicht beenden, bis alle Anwendungen, die Hilfe angefordert haben, *WinHelp* mit auf *Help_Quit* gesetztem *Command* aufgerufen haben.

WriteComm

Syntax
```
function WriteComm(Cid: Integer; Buf: PChar; Size:
Integer): Integer;
```

Beschreibung Diese Funktion schreibt die Anzahl der im Parameter *Size* angegebenen Zeichen zu der durch *Cid* bezeichneten Schnittstelle. Die Zeichen werden aus dem Datenpuffer entnommen, auf den der Parameter *Buf* weist.

Cid: Bestimmt die Schnittstelle, die die Zeichen empfangen soll. Die Funktion *OpenComm* gibt diesen Wert zurück.

Buf: Weist auf den Datenpuffer, der die zu schreibenden Zeichen enthält.

Size: Bestimmt die Zahl der zu schreibenden Zeichen.

Rückgabewert Der Rückgabewert gibt die Zahl der gerade geschriebenen Zeichen an. Wenn ein Fehler auftritt, wird der Rückgabewert auf einen Wert kleiner als 0 gesetzt, wobei der Absolutbetrag des Rückgabewerts die Zahl der tatsächlich geschriebenen Zeichen angibt. Die Fehlerursache kann mit der Funktion *GetCommError* bestimmt werden, die Fehlercode und -status liefert. Eine Übersicht der Fehlercodes finden Sie in Kapitel 1 »Windows-Stile und -Konstanten« im Abschnitt »ce_Xxxx«.

Hinweis *WriteComm* löscht Daten in der Übertragungs-Warteschlange, wenn dort nicht genügend Platz für zusätzliche Zeichen ist. Anwendungen sollten vor dem Aufruf von *WriteComm* den verfügbaren Platz in der Übertragungs-Warteschlange mit der Funktion *GetCommError* überprüfen. Außerdem sollten sie die Funktion *OpenComm* benutzen, um die Größe der Übertragungs-Warteschlange auf einen Umfang festzulegen, der nicht kleiner als die Größe des längsten erwarteteten Ausgabestrings ist.

WritePrivateProfileString

Syntax
```
function WritePrivateProfileString(ApplicationName,
KeyName, Str, FileName: PChar): Bool;
```

Beschreibung

Diese Funktion kopiert den durch den Parameter *Str* bezeichneten String in die angegebene Initialisierungsdatei. Sie sucht die Datei nach der Schlüsselanweisung ab, die durch den Parameter *KeyName* unter dem Programmkopf der Anwendung benannt ist, der wiederum durch den Parameter *ApplicationName* bestimmt ist. Wird keine Übereinstimmung gefunden, fügt sie dem Benutzerprofil einen neuen Stringeintrag hinzu, der die Bezeichnung der Schlüsselanweisung und den vom Parameter *Str* bestimmten Schlüsselwert enthält. Ist eine übereinstimmende Schlüsselanweisung vorhanden, ersetzt die Funktion den Wert dieser Schlüsselanweisung durch *Str*.

ApplicationName: Zeigt auf einen nullterminierten String, der den Programmkopf einer Anwendung in der Initialisierungsdatei enthält.

KeyName: Zeigt auf einen nullterminierten String, der eine Schlüsselbezeichnung enthält, die unter dem Programmkopf der Anwendung in der Initialisierungsdatei erscheint.

Str: Zeigt auf den nullterminierten String, der den neuen Wert der Schlüsselanweisung enthält.

FileName: Zeigt auf einen nullterminierten String, der die Initialisierungsdatei bezeichnet. Enthält *FileName* keinen vollständigen Pfadnamen für die Datei, durchsucht *WritePrivateProfileString* das Windows-Verzeichnis nach der Datei. Ist diese nicht vorhanden, wird sie dort erstellt. Enthält *FileName* den vollständigen Pfadnamen einer nicht existierenden Datei, so erstellt *WritePrivateProfileString* keine Datei.

Rückgabewert

Der Rückgabewert ist *True*, wenn die Funktion erfolgreich war, andernfalls ist er *False*.

Hinweis

Eine Anwendung sollte eine eigene (anwendungsspezifische) Initialisierungsdatei zum Aufzeichnen von Information, die nur diese Anwendung betrifft, benutzen. Dies erhöht die Ausführungsgeschwindigkeit der Anwendung und von Windows, indem es die Informationsmenge reduziert, die Windows beim Zugriff auf die Initialisierungsdatei lesen muß.

Ist kein Anwendungsfeld für *ApplicationName* vorhanden, erstellt *WritePrivateProfileString* ein neues Anwendungsfeld und plaziert eine entsprechende Schlüsselanweisung in diesem Feld der Initialisierungsdatei.

Ein Stringeintrag in der Initialisierungsdatei hat folgende Form:

```
[Anwendungsname]
Schlisselname = String
.
.
.
```

Eine Anwendung kann *WritePrivateProfileString* auch aufrufen, um Zeilen aus ihrer eigenen Initialisierungsdatei zu löschen. Ist *Str* **nil**, löscht die Funktion die ganze durch den Parameter *KeyName* bezeichnete Zeile. Weist *Str* auf einen Nullstring, löscht die Funktion nur den Wert, die Schlüsselbezeichnung bleibt in der Datei. Ist *KeyName* **nil**, löscht die Funktion den ganzen durch den Parameter *ApplicationName* bezeichneten Abschnitt, sie löscht jedoch keine Zeilen, die mit dem Kommentarzeichen Strichpunkt (;) beginnen.

WriteProfileString

Syntax

```
function WriteProfileString(ApplicationName, KeyName,
Str: PChar): Bool;
```

Beschreibung

Diese Funktion kopiert den String, auf den der Parameter *Str* zeigt, in die Windows-Initialisierungsdatei WIN.INI. Sie sucht WIN.INI nach der Schlüsselanweisung ab, die durch den Parameter *KeyName* unter dem Kopf des Anwendungsprogramms benannt wird. Das Anwendungsprogramm selbst wird durch den Parameter *ApplicationName* bezeichnet.

Wird keine Übereinstimmung gefunden, fügt *WriteProfileString* dem Benutzerprofil einen neuen Stringeintrag hinzu, der die Bezeichnung der Schlüsselanweisung und des Schlüsselwertes enthält. Ist eine entsprechende Schlüsselanweisung vorhanden, ersetzt die Funktion den Wert der Schlüsselanweisung durch *Str*.

ApplicationName: Zeigt auf einen nullterminierten String, der den Kopf eines Anwendungsprogramms in WIN.INI enthält.

KeyName: Zeigt auf einen nullterminierten String, der die Schlüsselbezeichnung enthält, die unter dem Kopf eines Anwendungsprogrammes in WIN.INI erscheint.

Str: Zeigt auf den nullterminierten String, der den neuen Wert der Schlüsselanweisung enthält.

Rückgabewert

Der Rückgabewert ist *True*, wenn die Funktion erfolgreich war, andernfalls ist er *False*.

Hinweis

Wenn für *ApplicationName* keine Entsprechung vorhanden ist, erzeugt diese Funktion ein neues Anwendungsfeld und fügt den String hinzu, auf den *Str* zeigt.

Ein Stringeintrag in WIN.INI hat folgende Form:

```
[Anwendungsname]
Schlisselname = String
.
.
.
```

Eine Anwendung kann *WriteProfileString* auch aufrufen, um Zeilen aus WIN.INI zu löschen.

Ist *Str* **nil**, löscht die Funktion die ganze durch den Parameter *KeyName* bezeichnete Zeile. Zeigt *Str* auf einen Nullstring, löscht die Funktion nur den Wert, die Schlüsselbezeichnung bleibt in der Datei.

Ist *KeyName* gleich **nil**, löscht die Funktion den ganzen vom Parameter *ApplikationName* bezeichneten Abschnitt, sie löscht jedoch keine Zeilen, die mit einem Strichpunkt (;) als Kommentarzeichen beginnen.

wvsprintf

Syntax

```
function wvsprintf(DestStr, Format: PChar; var
ArgList): Integer;
```

Beschreibung

Diese Funktion formatiert und speichert eine Folge von Zeichen in einen Puffer.

Die Elemente, auf die die Argumentliste zeigt, werden gemäß der zugehörigen Formatvorgabe konvertiert und ausgegeben. Die Funktion fügt eine 0 an das Ende der geschriebenen Zeichen, der Rückgabewert berücksichtigt das abschliessende Nullzeichen bei seiner Zeichenzählung nicht.

Output: Zeigt auf einen nullterminierten String, der die formatierte Ausgabe aufnehmen soll.

Format: Zeigt auf einen nullterminierten String, der die Formatvorgabe enthält. Zusätzlich zu den gewöhnlichen ASCII-Zeichen erscheint in dieser String eine Formatvorgabe für jedes Argument.

Arglist: Zeigt auf ein Argument-Array, in dem jedes Word ein Argument für die Formatvorgabe bezeichnet. Die Anzahl, der Typ und die Interpretation des Arguments hängen von den entsprechenden Zeichensequenzen der Formatangaben im Parameter *Format* ab.

- Jedes Zeichen oder jede ein Word umfassende Integerzahl (%c, %d, %x, %i) erfordert in *Arglist* ein Word.
- LongInt-Zahlen (%ld, %li, %lx) erfordern zwei Word, das niederwertige gefolgt vom höherwertigen Word.
- Ein String (%s) benötigt zwei Word, den Offset gefolgt vom Segment (die zusammen einen Zeiger ergeben).

Rückgabewert

Der Rückgabewert ist die Anzahl der in *Output* gespeicherten Zeichen, die abschließende 0 wird dabei nicht mitgezählt. Wenn ein Fehler auftritt, gibt die Funktion einen Wert zurück, der kleiner als die Länge von *Format* ist.

Yield

Syntax	`function Yield: Bool;`
Beschreibung	Diese Funktion unterbricht die aktuelle Task und startet beliebige wartende Tasks. *Yield* hat keinen Parameter.
Rückgabewert	Wird nicht verwendet.
Anmerkungen	Anwendungen, die mit Fenstern arbeiten, sollten besser eine *DispatchMessage*-, *PeekMessage*- oder *TranslateMessage*-Schleife einsetzen, als die Funktion *Yield* zu verwenden. Die *PeekMessage*-Schleife führt die Synchronisation von Botschaften genauer durch und unterbricht zur richtigen Zeit.

3. Windows-Botschaften

In diesem Kapitel werden alle Windows-Botschaften alphabetisch aufgeführt. Zu jeder Botschaft finden Sie eine Beschreibung ihrer Verwendungsmöglichkeit, ihrer Parameter sowie des Rückgabewertes, der von Windows erwartet oder zurückgeliefert wird. Die Parameter *wParam* und *lParam* gehören zu jeder Botschaft. Eine genaue Beschreibung des *TMsg*-Records mit den Feldern *wParam* und *lParam* finden sie in Kapitel 4 »Windows-Typen« im Abschnitt »TMsg-Typ«.

bm_GetCheck

Beschreibung	Diese Botschaft ermittelt, ob ein Auswahlfeld oder ein Schaltfeld markiert ist.
Parameter	*wParam:* Nicht verwendet.
	lParam: Nicht verwendet.
Rückgabewert	Der Rückgabewert ist ungleich 0, wenn das Auswahlfeld oder das Schaltfeld markiert ist, andernfalls ist er 0. Die Botschaft *bm_GetCheck* gibt für einen Schalter immer 0 zurück.

bm_GetState

Beschreibung	Diese Botschaft bestimmt den Status eines Schalters, wenn der Benutzer eine Maustaste oder die Leertaste drückt.
Parameter	*wParam:* Nicht verwendet.
	lParam: Nicht verwendet.
Rückgabewert	Die Botschaft *bm_GetState* gibt einen Wert ungleich 0 zurück, wenn eine der folgenden Bedingungen erfüllt ist, andernfalls ist der Rückgabewert 0:

- Ein Schalter ist hervorgehoben.
- Der Benutzer drückt eine Maustaste oder die Leertaste, wenn ein Schalter den Fokus hat.
- Der Benutzer drückt eine Maustaste, während der Mauszeiger auf einem Schalter steht.

bm_SetCheck

Beschreibung Diese Botschaft markiert ein Schaltfeld oder ein Auswahlfeld oder löscht die Auswahlmarkierung von einem Schaltfeld oder Auswahlfeld.

Parameter *wParam:* Bestimmt, ob eine Auswahlmarkierung für einen Schalter oder ein Auswahlfeld gesetzt oder gelöscht werden soll. Ist der Parameter *wParam* ungleich 0, wird eine Auswahlmarkierung gesetzt, ist er 0, wird sie entfernt.

Bei dreistufigen Schaltern wird eine Auswahlmarkierung neben den Schalter gesetzt, wenn *wParam* 1 ist. Ist *wParam* 2, wird der Schalter in den nicht aktivierbaren Zustand überführt und ist *wParam* 0, wird der Schalter in den normalen Zustand zurückgesetzt (keine Markierung, aber aktivierbar).

lParam: Nicht verwendet.

Rückgabewert Nicht verwendet.

Hinweis Die Botschaft *bm_SetCheck* hat keinen Einfluß auf Aktionsschalter.

bm_SetState

Beschreibung Diese Botschaft ändert den Status eines Schalters oder eines Auswahlfeldes.

Parameter *wParam:* Bestimmt die Art der Hervorhebung des Schalters. Ist der Parameter *wParam* ungleich 0, wird der Schalter hervorgehoben (das Innere wird invertiert dargestellt). Ist *wParam* 0, wird der Schalter in seinem regulären Status gezeichnet.

lParam: Nicht verwendet.

Rückgabewert Nicht verwendet.

Hinweis Aktionsschalter können nicht hervorgehoben werden.

bm_SetStyle

Beschreibung Diese Botschaft ändert den Stil eines Schalters. Weicht der Typ, der im Parameter *wParam* enthalten ist, vom vorgefundenen Stil ab, wird der Schalter in der neuen Form gezeichnet.

Parameter *wParam:* Gibt den Wert für den Stil an. Eine Beschreibung aller Schalterstile finden Sie in Kapitel 1 »Windows-Stile und -Konstanten« im Abschnitt »bs_Xxxx«.

lParam: Bestimmt, ob die Schalter neu gezeichnet werden sollen. Ist *lParam* 0, werden die Schalter nicht neu gezeichnet. Ist *lParam* ungleich 0, werden sie neu gezeichnet.

Rückgabewert Nicht verwendet.

cb_AddString

Beschreibung Diese Botschaft fügt einen String zur Liste eines Kombinationsfensters hinzu. Wenn die Liste nicht sortiert ist, wird der String am Ende der Liste angefügt, ansonsten wird der String nach erneutem Sortieren in die Liste eingefügt.

Diese Botschaft löscht jede vorhandene Listenauswahl.

Parameter *wParam:* Nicht verwendet.

lParam: Zeigt auf den nullterminierten String, der hinzugefügt werden soll. Wurde das Kombinationsfenster ohne Verwendung von *cbs_HasStrings* durch die Anwendung erstellt, ist der Parameter *lParam* ein von der Anwendung übergebener 32-Bit-Wert, der vom Kombinationsfenster anstelle des Zeigers auf den String gespeichert wird.

Rückgabewert Der Rückgabewert ist der Index zu String und Liste. Wenn ein Fehler auftritt, wird *cb_Err*, wenn zum Speichern des neuen Strings nicht genügend Speicher zur Verfügung steht, *cb_ErrSpace* zurückgegeben.

Hinweis Wird das Kombinationsfenster einer Anwendung mit *cbs_Sort* und ohne *cbs_HasStrings* erstellt , wird die Botschaft *wm_CompareItem* ein- oder mehrmals an den Besitzer des Kombinationsfensters gesendet, so daß der neue Eintrag richtig in der Liste angeordnet werden kann.

cb_DeleteString

Beschreibung Diese Botschaft löscht einen String aus der Liste eines Kombinationsfensters.

Parameter *wParam:* Enthält einen Index des Strings, der gelöscht werden soll.

lParam: Nicht verwendet.

Rückgabewert Der Rückgabewert ist die Anzahl der Strings, die in der Liste verbleiben. Der Rückgabewert ist *cb_Err,* wenn *wParam* keinen gültigen Index bezeichnet.

Hinweis Wenn das Kombinationsfenster anwendungsgezeichnet (*cbs_OwnerDrawFixed* oder *cbs_OwnerDrawVariable*) ist, aber ohne *cbs_HasStrings* erstellt wurde, wird eine Botschaft *wm_DeleteItem* an den Besitzer des Kombinationsfensters gesendet, so daß die Anwendung zusätzliche, dem Eintrag durch den Parameter *lParam* der Botschaften *cb_AddString* oder *cb_InsertString* zugeordnete Daten freigeben kann.

cb_Dir

Beschreibung Diese Botschaft fügt eine Liste der Dateien aus dem aktuellen Verzeichnis zur Liste hinzu. Es werden nur Dateien mit den von *wParam* vorgegebenen Attributen und der vom Parameter *lParam* gegebenen Dateispezifikation hinzugefügt.

Parameter *wParam:* Enthält einen DOS-Attributwert. Eine Liste der DOS-Attribute finden Sie in der Beschreibung der Funktion *DlgDirList*.

lParam: Zeigt auf einen Dateispezifikationsstring. Der String kann Platzhalterzeichen enthalten (zum Beispiel ***.***).

Rückgabewert Der Rückgabewert ist der Index des letzten Eintrags in der Liste. Der Rückgabewert ist *cb_Err,* wenn ein Fehler auftritt und *cb_ErrSpace,* wenn zum Speichern des neuen Strings nicht genügend Speicherplatz vorhanden ist.

Hinweis Der Rückgabewert der Botschaft *cb_Dir* ist um 1 kleiner, als der Rückgabewert der Botschaft *cb_GetCount*.

cb_FindString

Beschreibung Diese Botschaft findet den ersten String in der Liste eines Kombinationsfensters, der mit dem angegebenen Präfixtext übereinstimmt.

Parameter *wParam:* Enthält den Index zu dem Eintrag vor dem ersten zu suchenden Eintrag. Wenn die Suche das Ende einer Liste erreicht, wird sie vom Listenbeginn an bis zu dem von *wParam* bezeichneten Element fortgesetzt. Ist der Parameter *wParam* -1, wird die ganze Liste von Beginn an durchsucht.

lParam: Zeigt auf den Präfixstring. Der String muß nullterminiert sein.

Rückgabewert Der Rückgabewert ist der Index des ersten übereinstimmenden Eintrags oder *cb_Err,* wenn die Suche nicht erfolgreich war.

Hinweis Wenn das Kombinationsfenster anwendungsgezeichnet ist, aber ohne *cbs_HasStrings* erstellt wurde, gibt diese Botschaft den Index des Eintrags zurück, dessen zugehörige 32-Bit-Wert (übergeben als Parameter *lParam* der Botschaften *cb_AddString* oder *cb_InsertString)* mit dem Wert übereinstimmt, der als Parameter *lParam* von *cb_FindString* übergeben wurde.

cb_GetCount

Beschreibung Diese Botschaft gibt die Anzahl der Einträge in der Liste eines Kombinationsfensters zurück.

Parameter *wParam:* Nicht verwendet.

lParam: Nicht verwendet.

Rückgabewert Der Rückgabewert ist die Anzahl der Einträge in der Liste eines Kombinationsfensters.

cb_GetCurSel

Beschreibung Diese Botschaft gibt den Index des aktuell ausgewählten Eintrags in der Liste eines Kombinationsfensters zurück, falls ein solcher existiert.

Parameter *wParam:* Nicht verwendet.

lParam: Nicht verwendet.

Rückgabewert Der Rückgabewert ist der Index des aktuell ausgewählten Eintrags. Er ist *cb_Err,* wenn kein Eintrag ausgewählt ist.

cb_GetEditSel

Beschreibung Diese Botschaft gibt die Start- und Endpositionen des ausgewählten Texts im Editierfeld eines Kombinationsfensters zurück.

Parameter *wParam:* Nicht verwendet.

lParam: Nicht verwendet.

Rückgabewert Der Rückgabewert ist ein LongInt-Wert, der die Startposition im niederwertigen Word und die Endposition im höherwertigen Word angibt. Wenn die Botschaft an ein Kombinationsfenster ohne Editierfeld gesendet wurde, ist der Rückgabewert *cb_Err*.

cb_GetItemData

Beschreibung Diese Botschaft ruft den von der Anwendung bereitgestellten 32-Bit-Wert ab, der dem angegebenen Kombinationsfenstereintrag zugeordnet ist.

Wenn sich der Eintrag in einem anwendungsgezeichneten Kombinationsfenster befindet, das ohne *cbs_HasStrings* erstellt wurde, war dieser 32-Bit-Wert im Parameter *lParam* der Botschaften *cb_AddString* oder *cb_InsertString* enthalten, die den Eintrag zum Kombinationsfenster hinzufügten.

Andernfalls ist es der Wert des Parameters *lParam* einer Botschaft *cb_SetItemData*.

Parameter *wParam:* Enthält den Index eines Eintrags.

lParam: Nicht verwendet.

Rückgabewert Der Rückgabewert ist der 32-Bit-Wert, der dem Eintrag zugeordnet ist oder *cb_Err,* wenn ein Fehler auftritt.

cb_GetLBText

Beschreibung Diese Botschaft kopiert einen String aus der Liste eines Kombinationsfensters in einen Puffer.

Parameter *wParam:* Enthält den Index des Strings, der kopiert werden soll.

lParam: Zeigt auf einen Puffer, der den String aufnehmen soll. Der Puffer muß ausreichend Speicherplatz für den String und ein abschließendes Nullzeichen haben.

Rückgabewert Der Rückgabewert ist die Länge des Strings in Bytes, ohne Berücksichtigung des abschließenden Nullzeichens. Bezeichnet *wParam* keinen gültigen Index, ist der Rückgabewert *cb_Err.*

Hinweis Wurde das Kombinationsfenster mit *cbs_OwnerDrawFixed* oder *cbs_OwnerDrawVariable*, jedoch ohne *cbs_HasStrings* erstellt, so empfängt der Puffer, auf den der Parameter *lParam* zeigt, den 32-Bit-Wert, der dem Eintrag durch den Parameter *lParam* der Botschaften *cb_AddString* oder *cb_InsertString* zugeordnet wurde.

cb_GetLBTextLen

Beschreibung Diese Botschaft gibt die Länge eines Strings in der Liste eines Kombinationsfensters in Bytes zurück.

Parameter *wParam:* Enthält den Index des Strings.

lParam: Nicht verwendet.

Rückgabewert Der Rückgabewert ist die Länge des Strings in Bytes, ohne Berücksichtigung des abschließenden Nullzeichens. Wenn der Parameter *wParam* keinen gültigen Index angibt, ist der Rückgabewert *cb_Err.*

cb_InsertString

Beschreibung Diese Botschaft fügt einen String in die Liste eines Kombinationsfensters ein. Die Liste wird nicht sortiert.

Parameter *wParam:* Enthält den Index der Position, die den String aufnehmen soll. Wenn der Parameter *wParam* -1 ist, wird der String am Ende der Liste angefügt.

lParam: Zeigt auf den nullterminierten String, der eingefügt werden soll. Wenn das Kombinationsfenster ohne den Stil *cbs_HasStrings* erstellt wurde, ist der Parameter *lParam* ein

anwendungsunterstützter 32-Bit-Wert, der vom Kombinationsfenster anstelle des Zeigers auf den String gespeichert wird.

Rückgabewert Der Rückgabewert ist der Index der Position, an der der String eingefügt wurde.

Wenn ein Fehler auftritt, ist der Rückgabewert *cb_Err*, wenn zum Speichern des neuen Strings nicht genügend Speicherplatz zur Verfügung steht, ist er *cb_ErrSpace*.

cb_LimitText

Beschreibung Diese Botschaft begrenzt die Länge (in Bytes) des Textes, den der Benutzer in das Editierfeld eines Kombinationsfensters eingeben kann.

Parameter *wParam:* Bestimmt die Anzahl von Bytes, die der Benutzer höchstens eingeben kann. Is *wParam* 0, gibt es keine Beschränkung.

lParam: Nicht verwendet.

Rückgabewert Der Rückgabewert ist ungleich 0, wenn die Botschaft erfolgreich war, andernfalls ist er 0.

Wird diese Botschaft an ein Kombinationsfenster ohne Editierfeld gesendet, ist der Rückgabewert *cb_Err.*

cb_ResetContent

Beschreibung Diese Botschaft löscht alle Strings aus der Liste eines Kombinationsfensters und gibt den für diese Strings reservierten Speicherplatz frei.

Parameter *wParam:* Nicht verwendet.

lParam: Nicht verwendet.

Rückgabewert Nicht verwendet

Hinweis Wenn das Kombinationsfenster ohne *cbs_HasStrings*, aber mit *cbs_OwnerDrawFixed* oder *cbs_OwnerDrawVariable* erstellt wurde, erhält der Besitzer des Kombinationsfensters eine *wm_DeleteItem*-Botschaft für jeden Eintrag im Kombinationsfenster.

cb_SelectString

Beschreibung Diese Botschaft wählt den ersten String in der Liste eines Kombinationsfensters aus, der mit dem angegebenen Präfix übereinstimmt. Der Text im Editierfeld des Kombinationsfensters wird verändert, um die neue Auswahl widerzuspiegeln.

Parameter *wParam:* Enthält den Index zu dem Eintrag vor dem ersten Eintrag, ab dem gesucht werden soll. Wenn die Suche das Ende der Liste erreicht, wird sie vom Listenbeginn an bis zu dem von *wParam* bezeichneten Eintrag fortgeführt. Wenn *wParam* -1 ist, wird die ganze Liste von Beginn an durchsucht.

lParam: Zeigt auf den Präfix-String. Dieser muß ein abschliessendes Nullzeichen enthalten.

Rückgabewert Der Rückgabewert ist der Index des ersten übereinstimmenden Eintrags. Wenn die Suche nicht erfolgreich war, ist der Rückgabewert *cb_Err* und die aktuelle Auswahl wird nicht verändert.

Hinweis Ein String wird nur ausgewählt, wenn seine Anfangsbuchstaben (vom Startpunkt an) mit den Zeichen im Präfixstring übereinstimmen.

Wenn das Kombinationsfenster ohne *cbs_HasStrings* erstellt wurde, gibt diese Botschaft den Index zu dem Eintrag zurück, dessen zugehöriger 32-Bit-Wert (übergeben als Parameter *lParam* der Botschaften *cb_AddString* oder *cb_InsertString)* mit dem Wert übereinstimmt, der als Parameter *lParam* von *cb_FindString* übergeben wurde.

cb_SetCurSel

Beschreibung Diese Botschaft wählt einen String aus der Liste eines Kombinationsfensters aus und rollt ihn in den sichtbaren Bereich, wenn die Liste angezeigt ist. Außerdem wird der Text im Editierfeld oder im statischen Textfeld des Kombinationsfensters geändert, um die neue Auswahl widerzuspiegeln. Wenn der neue String ausgewählt ist, entfernt die Liste die Hervorhebung des vorher ausgewählten String.

Parameter *wParam:* Enthält den Index des Strings, der ausgewählt werden soll. Wenn *wParam* -1 ist, wird der Liste keine Auswahl zugewiesen.

lParam: Nicht verwendet.

Rückgabewert Wenn der von *wParam* bezeichnete Index ungültig ist, ist der Rückgabewert *cb_Err* und die aktuelle Auswahl wird nicht verändert.

cb_SetEditSel

Beschreibung Diese Botschaft wählt alle Zeichen im Editierfeld eines Kombinationsfensters aus, die innerhalb der Start- und Endpositionen liegen, die durch den Parameter *lParam* bestimmt werden.

Parameter *wParam:* Nicht verwendet.

lParamLo: Zeichenindex der Starposition.

lParamHi: Zeichenindex der Endposition.

Rückgabewert Der Rückgabewert ist ungleich 0, wenn die Botschaft erfolgreich war, andernfalls ist er 0.

Wenn die Botschaft an ein Kombinationsfenster ohne Editierfeld gesendet wird, ist der Rückgabewert *cb_Err*.

cb_SetItemData

Beschreibung Diese Botschaft setzt den 32-Bit-Wert, der dem angegebenen Eintrag in einem Kombinationsfenster zugeordnet ist.

Wenn sich der Eintrag in einem anwendungsgezeichneten Kombinationsfenster befindet, das ohne *cbs_HasStrings* erstellt wurde, ersetzt diese Botschaft den 32-Bit-Wert, der im Parameter *lParam* der Botschaften *cb_AddString* oder *cb_InsertString* enthalten war, die den Eintrag zum Kombinationsfenster hinzugefügt hatten.

Parameter *wParam:* Enthält den Index des Eintrags.

lParam: Enthält den neuen 32-Bit-Wert, der dem Eintrag zugeordnet werden soll.

Rückgabewert Der Rückgabewert ist *cb_Err,* wenn ein Fehler auftritt.

cb_ShowDropDown

Beschreibung Diese Botschaft zeigt die Drop-Down-Liste eines Kombinationsfenster, das mit dem Stil *cbs_DropDown* oder *cbs_DropDownList* erstellt wurde bzw. blendet sie aus.

Parameter *wParam:* Ist dieser Paramter ungleich 0, wird die Liste dargestellt, wenn sie nicht sichtbar ist. Ist er 0, wird die Liste ausgeblendet, wenn sie sichtbar ist.

lParam: Nicht verwendet.

Rückgabewert Nicht verwendet.

dm_GetDefID

Beschreibung Diese Botschaft ruft die ID des als Vorgabewert eingestellten Aktionsschalters für ein Dialogfenster ab.

Parameter *wParam:* Nicht verwendet.

lParam: Nicht verwendet.

Rückgabewert Der Rückgabewert ist ein 32-Bit-Wert. Das höherwertige Word enthält *dc_HasDefID*, falls ein Vorgabeschalter existiert,

andernfalls ist es 0. Das niederwertige Word enthält die ID des Vorgabeschalters, wenn das höherwertige Word *dc_HasDefID* enthält; andernfalls ist es 0.

dm_SetDefID

Beschreibung	Diese Botschaft wird von einer Anwendung benutzt, um die ID für den als Vorgabewert vorgesehenen Aktionsschalter eines Dialogfensters zu ändern.
Parameter	*wParam:* Enthält die neue ID des Vorgabeschalters.
	lParam: Nicht verwendet.
Rückgabewert	Nicht verwendet.

em_CanUndo

Beschreibung	Diese Botschaft bestimmt, ob ein Editierfeld korrekt auf eine *em_Undo*-Botschaft reagieren kann.
Parameter	*wParam:* Nicht verwendet.
	lParam: Nicht verwendet.
Rückgabewert	Der Rückgabewert ist ungleich 0, wenn das Editierfeld auf die Botschaft *em_CanUndo* korrekt reagieren kann, andernfalls 0.

em_EmptyUndoBuffer

Beschreibung	Diese Botschaft weist ein Editierfeld an, seinen Widerrufen-Puffer zu löschen. Dies nimmt dem Editierfeld die Fähigkeit, den letzten Editiervorgang rückgängig zu machen.
Parameter	*wParam:* Nicht verwendet.
	lParam: Nicht verwendet.
Rückgabewert	Nicht verwendet.
Hinweis	Der Puffer zum Widerrufen wird automatisch geleert, wenn das Editierfeld eine Botschaft *wm_SetText* oder *em_SetHandle* empfängt.

em_FmtLines

Beschreibung	Diese Botschaft weist ein Mehrzeilen-Editierfeld an, das Zeilenendezeichen aus umbrochenen Zeilen zu entfernen oder hinzuzufügen.
Parameter	*wParam:* Bestimmt die Verwendung von Zeilenendezeichen. Ist der Parameter *wParam* ungleich 0, werden die Zeichen CR

CR LF (0D 0D 0A hexadezimal) an das Ende der umbrochenen Zeilen gesetzt, ist er 0, werden die Zeilenendezeichen aus dem Text gelöscht.

lParam: Nicht verwendet.

Rückgabewert Der Rückgabewert ist ungleich 0, wenn der Text verändert wurde, andernfalls ist er 0.

Hinweis Zeilen, die mit einem Hard-Return enden (einem Wagenrücklaufzeichen, das vom Benutzer eingegeben wurde), erhalten die Zeichen CR LF am Zeilenende. Diese Zeilen werden von der Botschaft *em_FmtLines* nicht bearbeitet.

Beachten Sie, daß sich der Umfang des Textes ändert, wenn diese Botschaft verarbeitet wird.

em_GetHandle

Beschreibung Diese Botschaft gibt das Daten-Handle des Puffers zurück, der den Inhalt des Editierfeldes enthält. Das Handle ist immer ein lokales Handle auf eine Position im Datensegment der Anwendung.

Parameter *wParam:* Nicht verwendet.

lParam: Nicht verwendet.

Rückgabewert Der Rückgabewert ist das Daten-Handle des Puffers, der den Inhalt des Editierfeldes enthält.

Hinweis Eine Anwendung kann diese Botschaft nur an Editierfelder übergeben, deren Dialogfenster mit gesetztem Flag *ds_Local-Edit* erzeugt wurde.

em_GetLine

Beschreibung Diese Botschaft kopiert eine Zeile aus einem Editierfeld.

Parameter *wParam:* Bestimmt die Zeilennummer der Zeile im Steuerelement, wobei die Zeilennummer der ersten Zeile 0 ist.

lParam: Zeigt auf den Puffer, in dem die Zeile gespeichert werden soll.

Das erste Word des Puffers gibt die maximale Anzahl von Bytes an, die in den Puffer kopiert werden sollen. Die kopierte Zeile ist nicht mit 0 abgeschlossen.

Rückgabewert Der Rückgabewert ist die Anzahl von Bytes, die tatsächlich kopiert wurden.

Diese Botschaft wird von Einzeileneditierfeldern nicht bearbeitet.

em_GetLineCount

Beschreibung	Diese Botschaft gibt die Anzahl der Textzeilen eines Editierfelds zurück.
Parameter	*wParam:* Nicht verwendet.
	lParam: Nicht verwendet.
Rückgabewert	Der Rückgabewert ist die Anzahl der Textzeilen im Steuerelement.
Hinweis	Diese Botschaft wird von Einzeileneditierfeldern nicht verarbeitet.

em_GetModify

Beschreibung	Diese Botschaft gibt den aktuellen Wert des Veränderungsflags für ein gegebenes Editierfeld zurück. Das Flag wird vom Editierfeld gesetzt, wenn der Benutzer Text innerhalb des Editierfeldes eingibt oder bearbeitet.
Parameter	*wParam:* Nicht verwendet.
	lParam: Nicht verwendet.
Rückgabewert	Der Rückgabewert ist der Wert des aktuellen Veränderungsflags für ein vorgegebenes Editierfeld.

em_GetRect

Beschreibung	Diese Botschaft ruft das formatierende Rechteck eines Editierfeldes ab.
Parameter	*wParam:* Nicht verwendet.
	lParam: Zeigt auf eine *TRect*-Datenstruktur, die durch diese Botschaft gefüllt wird.
Rückgabewert	Nicht verwendet.

em_GetSel

Beschreibung	Diese Botschaft gibt die Position der Start- und Endzeichen der aktuellen Auswahl in einem Editierfeld zurück.
Parameter	*wParam:* Nicht verwendet.
	lParam: Nicht verwendet.
Rückgabewert	Der Rückgabewert enthält die Startposition im niederwertigen Word. Im höherwertigen Word erhält er die Position des ersten nicht ausgewählten Zeichens nach dem Ende der Auswahl.

em_LimitText

Beschreibung Diese Botschaft begrenzt die Anzahl der Zeichen, die der Anwender in ein Editierfeld eingeben kann.

Parameter *wParam:* Bestimmt die maximale Anzahl von Bytes, die eingegeben werden können. Wenn der Benutzer versucht, mehr Zeichen einzugeben, wird ein Warnton ausgegeben und die Zeichen werden nicht angenommen. Ist der Parameter *wParam* 0, wird die Textlänge nur durch den verfügbaren Speicherplatz begrenzt.

lParam: Nicht verwendet.

Rückgabewert Tritt ein Fehler auf, so ist der Rückgabewert 0, andernfalls ungleich 0.

Hinweis Die Botschaft *em_LimitText* betrifft keinen Text, der von der Botschaf *wm_SetText* gesetzt wurde, und keinen Puffer, der von der Botschaft *em_SetHandle* eingerichtet wurde.

em_LineFromChar

Beschreibung Diese Botschaft gibt die Zeilennummer der Zeile zurück, die das Zeichen enthält, dessen Index vom Parameter *wParam* vorgegeben ist.

Parameter *wParam:* Enthält den Indexwert für das gewünschte Zeichen im Text des Editierfeldes (diese Indexwerte beginnen mit 0) oder -1.

lParam: Nicht verwendet.

Rückgabewert Der Rückgabewert ist eine Zeilennummer. Ist *wParam* -1, wird die Nummer der Zeile zurückgegeben, die das erste Zeichen der Auswahl enthält. Andernfalls enthält *wParam* den Index (oder die Position) des gewünschten Zeichens im Text des Editierfeldes und es wird die Nummer der Zeile, die das Zeichen enthält, zurückgegeben.

em_LineIndex

Beschreibung Diese Botschaft gibt die Anzahl von Zeichenpositionen zurück, die vor dem ersten Zeichen einer angegebenen Zeile auftreten.

Parameter *wParam:* Bezeichnet die gewünschte Zeilennummer, wobei die Zeilennummer der ersten Zeile 0 ist. Ist der Parameter *wParam* -1, wird die aktuelle Zeilennummer (die Zeile, die das Caret enthält) verwendet.

lParam: Nicht verwendet.

Rückgabewert Der Rückgabewert ist die Anzahl von Zeichenpositionen, die dem ersten Zeichen in der vorgegebenen Zeile vorangehen.

Hinweis Diese Botschaft wird von Einzeileneditierfeldern nicht bearbeitet.

em_LineLength

Beschreibung Diese Botschaft gibt die Länge einer Zeile im Textpuffer eines Editierfeldes in Bytes zurück, die durch einen Zeichenindex bestimmt ist.

Parameter *wParam:* Bestimmt den Zeichenindex eines Zeichens in der angegebenen Zeile, wobei die Zeilennummer der ersten Zeile 0 ist.

lParam: Nicht verwendet.

Rückgabewert Ist der Parameter *wParam* -1, wird die Länge der aktuellen Zeile (der Zeile, die das Caret enthält), zurückgegeben – ohne Berücksichtigung eines ausgewählten Texts.

Umfaßt die aktuelle Auswahl mehr als eine Zeile, wird die gesamte Länge der Zeilen abzüglich des ausgewählten Texts zurückgegeben.

Hinweis Man verwendet die Botschaft *em_LineIndex,* um einen Zeichenindex für eine angegebene Zeilenummer abzurufen. Dieser Index kann mit der Botschaft *em_LineLength* benutzt werden.

em_LineScroll

Beschreibung Diese Botschaft rollt den Inhalt eines Editierfeldes.

Parameter *wParam:* Nicht verwendet.

lParamLo: Ist die Anzahl der Zeilen, die vertikal gerollt werden soll.

lParamHi: Ist die Anzahl der Spalten, um die horizontal gerollt werden soll.

Rückgabewert Nicht verwendet.

Hinweis Diese Botschaft wird von Einzeileneditierfeldern nicht bearbeitet.

em_ReplaceSel

Beschreibung Diese Botschaft ersetzt die aktuelle Auswahl im Editierfeld durch neuen Text.

Parameter *wParam:* Nicht verwendet.

lParam: Zeigt auf einen nullterminierten String, der den Ersatztext enthält.

Rückgabewert Nicht verwendet.

em_SetHandle

Beschreibung Diese Botschaft richtet den Textpuffer ein, der benutzt wird, um den Inhalt eines Editierfeldes zu speichern.

Parameter *wParam:* Enthält ein Handle des Puffers. Das Handle muß ein lokales Handle zu einer Position im Datensegment der Anwendung sein. Das Editierfeld verwendet diesen Puffer, um den aktuell angezeigten Text zu speichern, anstatt einen eigenen Puffer zu reservieren. Falls nötig, belegt das Steuerelement diesen Puffer neu.

lParam: Nicht verwendet.

Rückgabewert Nicht verwendet.

Hinweis Diese Botschaft wird von Einzeileneditierfeldern nicht verarbeitet. Wenn *em_SetHandle* benutzt wird, um den von einem Editierfeld benutzten Textpuffer zu verändern, wird der vorherige Textpuffer nicht entfernt. Die Anwendung muß das vorherige Puffer-Handle mit Hilfe der Botschaft *em_GetHandle* abrufen und mit der Funktion *LocalFree* freigeben.

Ein Editierfeld belegt den gegebenen Puffer immer dann neu, wenn es zusätzlichen Speicherplatz für Text benötigt oder wenn es genügend Text löscht, so daß zusätzlicher Speicherplatz nicht mehr gebraucht wird. Eine Anwendung kann diese Botschaft nur an solche Editierfelder übergeben, deren Dialogfenster mit gesetztem Modus-Flag *ds_LocalEdit* erzeugt wurden.

em_SetModify

Beschreibung Diese Botschaft setzt das Veränderungsflag für ein gegebenes Editierfeld.

Parameter *wParam:* Gibt einen neuen Wert für das Veränderungsflag an.

lParam: Nicht verwendet.

Rückgabewert Nicht verwendet.

em_SetPasswordChar

Beschreibung Diese Botschaft setzt das Zeichen, das in einem mit dem Stil *Password* erzeugten Editierfeld anstelle der eingegebenen Zeichen angezeigt wird.

Das voreingestellte Zeichen ist ein Stern (*).

Parameter *wParam:* Gibt das Zeichen an, das anstelle des tatsächlich vom Benutzer eingegebenen Zeichens dargestellt werden soll. Ist *wParam* 0, werden die tatsächlich vom Benutzer getippten Zeichen ausgegeben.

lParam: Nicht verwendet.

Rückgabewert Nicht verwendet.

em_SetRect

Beschreibung Diese Botschaft setzt das formatierende Rechteck für ein Editierfeld. Der Text wird neu formatiert und neu dargestellt, um dem veränderten Rechteck zu entsprechen.

Parameter *wParam:* Nicht verwendet.

lParam: Zeigt auf eine *TRect*-Datenstruktur, welche die neuen Abmessungen des Rechtecks vorgibt.

Rückgabewert Nicht verwendet.

Hinweis Diese Botschaft wird von Einzeileneditierfeldern nicht verarbeitet.

em_SetRectNP

Beschreibung Diese Botschaft setzt das formatierende Rechteck für ein Editierfeld. Der Text wird neu formatiert, um dem veränderten Rechteck zu entsprechen. Die Botschaft *em_SetRectNP* ähnelt der Botschaft *em_SetRect,* allerdings wird das Steuerelement nicht neu gezeichnet. Alle darauffolgenden Änderungen bewirken, daß das Steuerelement neu gezeichnet wird, um dem geänderten, formatierenden Rechteck zu entsprechen. Diese Botschaft wird benutzt, wenn das Feld erst nachträglich neu gezeichnet werden soll.

Parameter *wParam:* Nicht verwendet.

lParam: Zeigt auf eine *TRect*-Datenstruktur, welche die neuen Abmessungen des Rechtecks vorgibt.

Rückgabewert Nicht verwendet.

Hinweis Diese Botschaft wird von Einzeileneditierfeldern nicht bearbeitet.

em_SetSel

Beschreibung Diese Botschaft wählt alle Zeichen in einem Editierfeld aus, die innerhalb der Start- und Endzeichenpositionen liegen, die der Parameter *lParam* vorgibt.

Parameter *wParam:* Nicht verwendet.

lParamLo: Ist der Zeichenindex der Starposition.

lParamHi: Ist der Zeichenindex der Endposition.

Rückgabewert Nicht verwendet.

em_SetTabStops

Beschreibung Diese Botschaft setzt die Tabulatorpositionen in einem Mehrzeileneditierfeld.

Parameter *wParam:* Ist ein Word, das die Anzahl der Tabulatorpositionen im Editierfeld angibt. Es kann auch die Werte 0 oder 1 annehmen.

lParam: Ist ein Zeiger auf das erste Element eines Arrays von Integerwerten, die die Tabulatorpositionen in Dialogeinheiten enthalten. Eine Dialogeinheit ist ein horizontaler oder vertikaler Abstand. Eine horizontale Dialogeinheit ist gleich 1/4 und eine vertikale gleich 1/8 ihrer dazugehörigen Basiseinheit. Die Dialogbasiseinheiten werden aufgrund der Höhe und Breite der aktuellen Systemschrift gesetzt.

Die Funktion *GetDialogBaseUnits* gibt die aktuellen Dialogbasiseinheiten in Pixeln zurück. Die Tabulatorpositionen müssen in aufsteigender Reihenfolge sortiert werden, Rückwärtstabulatoren sind nicht erlaubt.

Rückgabewert Der Rückgabewert ist *True*, wenn alle Tabulatoren gesetzt wurden, andernfalls *False*.

Hinweis Sind *wParam* und *lParam* 0, werden voreingestellte Tabulatorpositionen im Abstand von 32 Dialogeinheiten gesetzt. Ist *wParam* 1, werden im Editierfeld Tabulatoren an jeder Position gesetzt, die einem Vielfachen von *lParam* entspricht.

em_SetWordBreak

Beschreibung Diese Botschaft wird an ein Mehrzeileneditierfeld gesendet, um dieses darüber zu informieren, daß Windows die voreingestellte Wortumbruchfunktion durch eine von der Anwendung übergebene Wortumbruchfunktion ersetzt hat.

Eine Wortumbruchfunktion durchsucht einen Textpuffer – der Text enthält, der an den Bildschirm gesendet werden soll – sucht nach dem ersten Wort, das nicht mehr in die aktuelle Bildschirmzeile paßt, und setzt dieses Wort an den Anfang der nächsten Zeile auf dem Bildschirm.

Eine Wortumbruchfunktion definiert, an welchem Punkt Windows eine Textzeile in Mehrzeileneditierfeldern trennen soll, üblicherweise ist dies bei einem Leerzeichen, das zwei

Worte trennt. Die von der Anwendung bereitgestellte Funktion kann einen Wortumbruch an einem Bindestrich oder einem anderen Zeichen als dem Leerzeichen definieren.

Parameter *wParam:* Nicht verwendet.

lParam: Die Adresse einer Prozedurinstanz. Diese Adresse muß mit der Funktion *MakeProcInstance* erzeugt werden. Die Wortumbruchfunktion kann folgendermaßen deklariert werden:

```
function WordBreakFunction (EditText: PChar;
CurrentWord: Integer; EditTextCount: Integer): PChar;
```

Sie können statt des Namens *WordBreakFunktion* auch jeden anderen Namen verwenden. Der Parameter *EditText* zeigt auf den Text eines Editierfeldes, der Parameter *CurrentWord* ist der Index des ersten Zeichens des aktuellen Worts. Der Parameter *EditTextCount* ist die Anzahl der Bytes des Textes.

Die Wortumbruchfunktion muß einen Zeiger auf den Angang des nächsten Wortes im Text zurückgeben. Ist das aktuelle Wort das letzte Wort im Text, sollte die Funktion einen Zeiger auf das Zeichen unmittelbar nach dem letzten Zeichen des Textes zurückgeben.

Rückgabewert Nicht verwendet.

Hinweis Die Standardwortumbruchfunktion von Windows definiert den Anfang des nächsten Worts als erstes Nicht-Leerzeichen nach einer Folge von Leerzeichen. Diese Botschaft ist nur auf Mehrzeileneditierfelder anwendbar.

em_Undo

Beschreibung Diese Botschaft macht den letzten Bearbeitungsvorgang im Editierfeld rückgängig. Verändert der Benutzer das Editierfeld, wird die letzte Änderung in einem Widerrufenpuffer gespeichert, der dynamisch mit den Anforderungen wächst. Ist für den Puffer nicht genügend Platz verfügbar, schlägt der Widerruf-Versuch fehl und das Editierfeld bleibt unverändert.

Parameter *wParam:* Nicht verwendet.

lParam: Nicht verwendet.

Rückgabewert Bei erfolgreicher Ausführung der Undo-Operation ist der Rückgabewert ungleich 0, andernfalls 0. In diesem Fall bleibt das Editierfeld unverändert.

lb_AddString

Beschreibung Diese Botschaft fügt einen String einer Liste hinzu. Ist die Liste unsortiert, wird der String am Ende der Liste angehängt, ist sie

sortiert, wird der String in die Liste eingeordnet. Diese Botschaft löscht alle vorhandenen Listenauswahlen.

Parameter *wParam:* Nicht verwendet.

lParam: Zeigt auf den nullterminierte String, der hinzugefügt werden soll. Wenn die Liste besitzergezeichnet ist, aber ohne den Stil *lbs_HasStrings* erzeugt wurde, ist *lParam* ein von der Anwendung bereitgestellter 32-Bit-Wert, der anstelle des Zeigers auf den String der Liste gespeichert wird.

Rückgabewert Der Rückgabewert ist der Index des Strings in der Liste. Der Rückgabewert ist *lb_Err,* wenn ein Fehler auftritt, und *lb_ErrSpace,* wenn nicht genug Speicherplatz zur Verfügung steht, um den neuen String zu speichern.

Hinweis Wenn eine besitzergezeichnete Liste mit dem Stil *lbs_Sort,* aber ohne *lbs_HasStrings* erstellt wurde, wird die Botschaft *nm_CompareItem* ein- oder mehrmals an den Besitzer der Liste gesendet, so daß der neue Eintrag korrekt in der Liste plaziert werden kann.

lb_DeleteString

Beschreibung Diese Botschaft löscht einen String aus der Liste.

Parameter *wParam:* Enthält einen Index des Strings, der gelöscht werden soll.

lParam: Nicht verwendet.

Rückgabewert Der Rückgabewert ist die Anzahl der verbleibenden Einträge in der Liste. Der Rückgabewert ist *lb_Err,* wenn ein Fehler auftritt.

Hinweis Wenn die Liste besitzergezeichnet ist, aber ohne den Stil *lbs_HasStrings* erstellt wurde, wird eine Botschaft *wm_DeleteItem* an den Eigentümer der Liste gesendet, so daß die Anwendung zusätzliche, dem Eintrag (durch den Parameter *lParam* der Botschaft *lb_AddString* oder *lb_InsertString*) zugeordnete Daten freigeben kann.

lb_Dir

Beschreibung Diese Botschaft fügt eine Liste der Dateien aus dem aktuellen Vezeichnis zur Liste hinzu. Es werden nur Dateien mit den vom Parameter *wParam* bezeichneten Attributen sowie Dateien, die der vom Parameter *lParam* gegebenen Dateivorgabe entsprechen, hinzugefügt.

Parameter *wParam:* Enthält einen DOS-Attributwert. Die DOS-Attribute werden bei der Funktion *DlgDirList* beschrieben.

lParam: Zeigt auf den nullterminierten Dateivorgabestring. Der String kann Stellvertreterzeichen enthalten (zum Beispiel *.*).

Rückgabewert Es wird der Index des letzten dargestellten Eintrags zurückgegeben. Der Rückgabewert ist *lb_Err,* wenn ein Fehler aufritt und *lb_ErrSpace,* wenn zum Speichern der neuen Strings nicht ausreichend Speicherplatz vorhanden ist.

Hinweis Der Rückgabewert der Botschaft *lb_Dir* ist um 1 kleiner als der Rückgabewert der Botschaft *lb_GetCount.*

lb_FindString

Beschreibung Diese Botschaft findet den ersten String in der Liste, der mit dem angegebenen Präfixtext übereinstimmt.

Parameter *wParam:* Enthält den Index des Eintrags vor dem ersten Eintrag, der untersucht werden soll. Wenn die Suche das Listenende erreicht, wird die Suche vom Beginn der Liste bis zu dem von *wParam* bezeichneten Eintrag fortgesetzt. Ist *wParam* -1, wird die gesamte Liste von Anfang an durchsucht.

lParam: Zeigt auf den Präfix-String. Der String muß mit 0 abgeschlossen sein.

Rückgabewert Der Rückgabewert ist der Index des ersten entsprechenden Eintrags oder *lb_Err*, wenn die Suche nicht erfolgreich war.

Hinweis Wenn die Liste besitzergezeichnet ist, aber ohne den Stil *lbs_HasStrings* erstellt wurde, gibt diese Botschaft den Index des Eintrags zurück, dessen zusätzlicher 32-Bit-Wert (übergeben als Parameter *lParam* der Botschaft *lb_AddString* oder *lb_InsertString*) mit dem Wert übereinstimmt, der von der Botschaft *lb_FindString* als Parameter *lParam* übergeben wurde.

lb_GetCount

Beschreibung Diese Botschaft gibt die Anzahl der Einträge in einer Liste zurück.

Parameter *wParam:* Nicht verwendet.

lParam: Nicht verwendet.

Rückgabewert Es wird die Anzahl der Einträge in der Liste zurückgegeben oder *lb_Err,* wenn ein Fehler auftritt.

lb_GetCursel

Beschreibung Diese Botschaft gibt den Index des aktuell ausgewählten Eintrags zurück, falls ein solcher existiert.

Parameter *wParam:* Nicht verwendet.

lParam: Nicht verwendet.

Rückgabewert Der Rückgabewert ist der Index des aktuell ausgewählten Eintrags. Er ist *lb_Err,* wenn kein Eintrag ausgewählt ist oder wenn die Liste für Mehrfachauswahl vorgesehen ist.

lb_GetHorizontalExtent

Beschreibung Diese Botschaft ruft die Breite in Pixeln ab, um die in der Liste ein Bildlauf durchgeführt werden kann, wenn die Liste horizontale Bildlaufleisten hat.

Parameter *wParam:* Nicht verwendet.

lParam: Nicht verwendet.

Rückgabewert Der Rückgabewert ist die Breite des Bildlaufbereichs in der Liste in Pixeln.

Hinweis Um auf die Botschaft *lb_GetHotizontalExtent* zu reagieren, muß die Liste mit dem Stil *ws_HScroll* definiert worden sein.

lb_GetItemData

Beschreibung Diese Botschaft ruft den von der Anwendung bereitgestellten 32-Bit-Wert ab, der dem angegebenen Listeneintrag zugeordnet ist. Befindet sich der Eintrag in einer besitzergezeichneten Liste, die ohne den Stil *lbs_HasStrings* erstellt wurde, war dieser 32-Bit-Wert im Parameter *lParam* der Botschaft *lb_AddString* oder *lb_InsertString* enthalten, die den Eintrag zur Liste hinzugefügt hat. Andernfalls war es der Wert im Parameter *lParam* einer Botschaft *lb_SetItemData*.

Parameter *wParam:* Enthält einen Index des Eintrags.

lParam: Nicht verwendet.

Rückgabewert Der Rückgabewert ist der 32-Bit-Wert, der dem Eintrag zugeordnet ist oder *lb_Err,* wenn ein Fehler auftritt.

lb_GetItemRect

Beschreibung Diese Botschaft ruft die Abmessungen des Rechtecks ab, das einen Listeneintrag so umgrenzt, wie er aktuell im Listenfenster dargestellt wird.

Parameter *wParam:* Enthält einen Index des Eintrags.

lParam: Enthält einen Zeiger auf eine *TRect*-Datenstruktur, die die Client-Koordinaten des Rechtecks aufnimmt.

Rückgabewert Der Rückgabewert ist *lb_Err,* wenn ein Fehler auftritt.

lb_GetSel

Beschreibung Diese Botschaft gibt den Auswahlstatus eines Eintrags zurück.

Parameter *wParam:* Enthält den Index des Eintrags.

lParam: Nicht verwendet.

Rückgabewert Der Rückgabewert ist eine positive Zahl, wenn ein Eintrag ausgewählt ist, andernfalls ist er 0. Der Rückgabewert ist *lb_Err,* wenn ein Fehler auftritt.

lb_GetSelCount

Beschreibung Diese Botschaft gibt die Gesamtzahl ausgewählter Einträge in einer Liste mit Mehrfachauswahlmöglichkeit zurück.

Parameter *wParam:* Nicht benutzt.

lParam: Nicht benutzt.

Rückgabewert Der Rückgabewert ist die Anzahl der ausgewählten Einträge in einer Liste.

Wenn die Liste nur für Einzelauswahl vorgesehen ist, ist der Rückgabewert *lb_Err*.

lb_GetSelItemS

Beschreibung Diese Botschaft füllt einen Puffer mit einem Array von Indizes der ausgewählten Einträgen in einer Liste mit Mehrfachauswahl.

Parameter *wParam:* Gibt die maximale Anzahl von ausgewählten Einträgen an, deren Indizes im Puffer gespeichert werden sollen.

lParam: Enthält einen Zeiger auf einen Puffer, der für die Anzahl von Integerwerten, die der Parameter *wParam* angibt, ausreichend groß sein muß.

Rückgabewert Der Rückgabewert ist die Anzahl der tatsächlich im Puffer gespeicherten Einträge.

Wenn die Liste eine Liste mit Einzelauswahl ist, ist der Rückgabewert *lb_Err*.

lb_GetText

Beschreibung Diese Botschaft kopiert einen String aus der Liste in einen Puffer.

Parameter *wParam:* Enthält den Index des Strings, der kopiert werden soll.

lParam: Zeigt auf den Puffer, der den String aufnehmen soll. Der Puffer muß groß genug sein, um sowohl den String als auch dessen abschließendes Null-Zeichen aufnehmen zu können.

Rückgabewert Der Rückgabewert ist die Länge des Strings (in Bytes) ohne Berücksichtigung des abschließenden Null-Zeichens.

Der Rückgabewert ist *lb_Err,* wenn der Parameter *wParam* keinen gültigen Index darstellt.

Hinweis Wenn die Liste anwendungsgezeichnet ist, aber ohne den Stil *lbs_HasStrings* erstellt wurde, empfängt der Puffer, auf den der Parameter *lParam* der Botschaft zeigt, den 32-Bit-Wert, der dem Eintrag durch den Parameter *lParam* der Botschaft *lb_AddString* oder *lb_InsertString* zugeordnet wurde.

lb_GetTextLen

Beschreibung Diese Botschaft gibt die Länge eines Strings in der Liste zurück.

Parameter *wParam:* Enthält einen Index des Eintrags.

lParam: Nicht verwendet.

Rückgabewert Der Rückgabewert ist die Länge des Strings in Bytes ohne Berücksichtigung des abschließenden Null-Zeichens. Der Rückgabewert ist *lb_Err,* wenn ein Fehler auftritt.

lb_GetTopIndex

Beschreibung Diese Botschaft gibt den Index des ersten sichtbaren Eintrags in einer Liste zurück. Zu Beginn steht Eintrag 0 ganz oben in der Liste. Wird aber in der Liste ein Bildlauf durchgeführt, kann auch ein anderer Eintrag an oberster Stelle stehen.

Parameter *wParam:* Nicht verwendet.

lParam: Nicht verwendet.

Rückgabewert Der Index des ersten sichtbaren Eintrags in einer Liste.

lb_InsertString

Beschreibung Diese Botschaft fügt einen String in eine Liste ein. Es wird keine Sortierung vorgenommen.

Parameter *wParam:* Enthält einen Index zu der Position, die den String aufnehmen soll. Wenn der Parameter *wParam* -1 ist, wird der String am Ende der Liste angefügt.

lParam: Zeigt auf den nullterminierte String, der eingefügt werden soll. Wenn die Liste besitzergezeichnet ist, aber ohne den Stil *lbs_HasStrings* erzeugt wurde, ist der Parameter *lParam* ein von der Anwendung unterstützter 32-Bit-Wert, der von der Liste anstelle des Zeigers auf den String gespeichert wird.

Rückgabewert Der Rückgabewert ist der Index der Position, an der der String eingefügt wurde. Es wird *lb_Err* zurückgegeben, wenn ein Fehler auftritt und *lb_ErrSpace,* wenn zum Abspeichern des neuen Strings nicht genügend Speicherplatz zur Verfügung steht.

lb_ResetContent

Beschreibung Diese Botschaft löscht alle Einträge aus einer Liste und gibt den gesamten für diese Strings reservierten Speicher frei.

Parameter *wParam:* Nicht verwendet.

lParam: Nicht verwendet.

Rückgabewert Nicht verwendet.

Hinweis Wenn die Liste besitzergezeichnet ist, aber ohne den Stil *lbs_HasStrings* erstellt wurde, empfängt der Besitzer der Liste eine Botschaft *wm_DeleteItem* für jeden Eintrag in der Liste.

lb_SelectStrings

Beschreibung Diese Botschaft setzt die aktuelle Auswahl auf den ersten String, der dem angegebenen Präfix entspricht.

Parameter *wParam:* Enthält den Index des Eintrags vor dem ersten zu untersuchenden Eintrag. Wenn die Suche das Listenende erreicht, wird sie vom Listenbeginn an bis zu dem von *wParam* bezeichneten Eintrag fortgesetzt. Wenn der Parameter *wParam* -1 ist, wird die gesamte Liste von Beginn an abgesucht.

lParam: Zeigt auf den Präfix-String. Der String muß ein abschließendes Null-Zeichen haben.

Rückgabewert Der Rückgabewert ist der Index des ersten entsprechenden Eintrags oder *lb_Err,* wenn ein Fehler auftritt.

Hinweis Diese Botschaft darf nicht für Listen verwendet werden, die für Mehrfachauswahl vorgesehen sind.

Ein String wird nur ausgewählt, wenn seine Anfangsbuchstaben den Zeichen im Präfix-String entsprechen.

Wenn die Liste besitzergezeichnet ist, aber ohne den Stil *lbs_HasStrings* erstellt wurde, gibt diese Botschaft den Index des Eintrags zurück, dessen zusätzlicher 32-Bit-Wert (übergeben als Parameter *lParam* der Botschaft *lb_AddString* oder

lb_InsertString) dem als Parameter *lParam* übergebenen Wert von *lb_FindString* entspricht.

lb_SelItemRange

Beschreibung	Diese Botschaft wählt einen oder mehrere aufeinanderfolgende Einträge in einer Mehrfachauswahlliste aus.
Parameter	*wParam:* Gibt an, wie die Auswahl gesetzt werden soll. Ist der Parameter *wParam* ungleich 0, werden die Einträge ausgewählt und hervorgehoben, ist *wParam* 0, wird die Hervorhebung zurückgenommen. Die Einträge sind dann nicht mehr ausgewählt. *lParamLo:* Der Index des ersten auszuwählenden Eintrags. *lParamHI*: Der Index des letzten auszuwählenden Eintrags.
Rückgabewert	Der Rückgabewert ist *lb_Err,* wenn ein Fehler auftritt.
Hinweis	Diese Botschaft sollte nur für Mehrfachauswahlfenster verwendet werden.

lb_SetColumnWidth

Beschreibung	Diese Botschaft wird an eine Mehrspaltenliste gesendet, die mit dem Stil *lbs_MultiColumn* erstellt wurde, um die Breite der Spalten in der Liste in Pixeln zu setzen.
Parameter	*wParam:* Gibt die Breite aller Spalten in Pixeln an. *lParam*: Nicht verwendet.
Rückgabewert	Nicht verwendet.

lb_SetCursel

Beschreibung	Diese Botschaft wählt einen Eintrag aus und läßt ihn ins Blickfeld rollen, falls dies notwendig ist. Wenn der neue Eintrag ausgewählt ist, löscht die Liste die Hervorhebung des vorher ausgewählten.
Parameter	*wParam:* Enthält den Index des Eintrags, der ausgewählt werden soll. Wenn *wParam* -1 ist, wird der Liste keine Auswahl zugeordnet. *lParam*: Nicht verwendet.
Rückgabewert	Der Rückgabewert ist *lb_Err,* wenn *wParam* -1 oder ein ungültiger Index ist, andernfalls wird der Index des ausgewählten Eintrags zurückgegeben.

Hinweis Diese Botschaft sollte nur in Einzelauswahlfenstern benutzt werden. Sie kann nicht eingesetzt werden, um eine Auswahl in einem Mehrfachauswahlfenster zu setzen oder um sie zu löschen.

lb_SetHorizontalExtent

Beschreibung Diese Botschaft stellt die Breite in Pixeln ein, um die ein horizontaler Bildlauf durchgeführt werden kann.

Ist die Liste schmaler als dieser Wert, kann horizontaler Bildlauf in der Liste durchgeführt werden. Wenn die Liste größer oder gleich diesem Wert ist, wird die horizontale Bildlaufleiste deaktiviert.

Parameter *wParam:* Bestimmt die Anzahl der Pixel, um die die Liste verschoben werden kann.

lParam: Nicht verwendet.

Rückgabewert Nicht verwendet.

Hinweis Um auf die Botschaft *lb_SetHorizontalExtent* reagieren zu können, muß die Liste mit dem Stil *ws_HScroll* erstellt worden sein.

lb_SetItemData

Beschreibung Diese Botschaft setzt einen 32-Bit-Wert, der dem angegebenen Eintrag in einer Liste zugeordnet ist. Wenn der Eintrag in einer besitzergezeichneten Liste steht, die ohne den Stil *lbs_HasStrings* erstellt wurde, ersetzt diese Botschaft den 32-Bit-Wert, der im Parameter *lParam* der Botschaft *lb_AddStrings* oder *lb_InsertString,* die den Eintrag zu der Liste hinzugefügt hat, enthalten war.

Parameter *wParam:* Enthält einen Index des Eintrags.

lParam: Enthält den neuen 32-Bit-Wert, der dem Eintrag zugeordnet werden soll.

Rückgabewert Der Rückgabewert ist *lb_Err,* wenn ein Fehler auftritt.

lb_SetSel

Beschreibung Diese Botschaft wählt in einer Liste mit Mehrfachauswahl einen Eintrag aus oder hebt dessen Auswahl auf.

Parameter *wParam:* Bestimmt, wie die Auswahl gesetzt werden soll. Ist der Parameter *wParam* ungleich 0, wird der Eintrag ausgewählt und hervorgehoben; ist er 0, wird die Hervorhebung zurückgenommen und der String ist nicht mehr ausgewählt.

lParam: Ist *lParam* -1, bezieht sich diese Botschaft auf alle Listeneinträge, andernfalls enthält *lParamLo* den Index des zu bearbeitenden Eintrags.

LParamLo: Der Index des Eintrags, wenn *lParam* nicht -1 ist.

Rückgabewert Der Rückgabewert ist *lb_Err,* wenn ein Fehler auftritt.

Hinweis Diese Botschaft sollte nur für Mehrfachauswahllisten verwendet werden.

lb_SetTabStops

Beschreibung Diese Botschaft setzt die Tabulatoren in einer Liste.

Parameter *wParam:* Ein Integer, der die Anzahl der Tabulatoren in der Liste angibt.

lParam: Ein Zeiger auf das erste Element in einem Array aus Integerwerten, das die Tabulatorpositionen in Dialogeinheiten enthält. Eine Dialogeinheit ist ein horizontaler oder vertikaler Abstand. Eine horizontale Dialogeinheit ist gleich einem Viertel der zugehörigen Basiseinheit.

Bei der Berechnung der Basisdialogeinheiten werden Breite und Höhe der aktuellen Systemschrift zugrundegelegt. Die Funktion *GetDialogBaseUnits* gibt die aktuellen Dialogbasiseinheiten in Pixeln zurück. Die Tabulatoren müssen in aufsteigender Ordnung sortiert werden, Rückwärtstabulatoren sind nicht zulässig.

Rückgabewert Der Rückgabewert ist ungleich 0, wenn alle Tabulatoren gesetzt wurden, ansonsten ist er 0.

Hinweis Wenn *wParam* und *lParam* 0 sind, beträgt der voreingestellte Tabulatorsprung zwei Dialogeinheiten. Wenn *wParam* 1 ist, verfügt das Editierfeld über Tabulatoren an jeder Position, die einem Vielfachen von *lParam* entspricht. Wenn *lParam* auf mehr als einen Wert zeigt, wird für jeden Wert in *lParam* ein Tabulator gesetzt, bis zu der Anzahl, die *wParam* vorgibt. Um auf die Botschaft *lb_SetTabStops* reagieren zu können, muß die Liste mit dem Stil *lbs_UseTabStops* erstellt worden sein.

lb_SetTopIndex

Beschreibung Diese Botschaft macht das durch den Index bezeichnete Element zum ersten sichtbaren Eintrag in einer Liste.

Parameter *wParam:* Gibt den Index des Listeneintrags an.

lParam: Nicht verwendet.

Rückgabewert Der Rückgabewert ist *lb_Err,* wenn ein Fehler auftritt.

wm_Activate

Beschreibung Diese Botschaft wird gesendet, wenn ein Fenster aktiviert oder deaktiviert wird.

Parameter *wParam:* Bestimmt den neuen Status des Fensters. Der Parameter *wParam* ist 0, wenn das Fenster nicht aktiv ist, andernfalls hat er einen der folgenden Werte:

Wert	Bedeutung
1	Das Fenster wurde nicht durch einen Mausklick, sondern zum Beispiel durch einen Aufruf der Funktion *SetActiveWindow* oder durch Auswahl des Anwenders über die Tastaturschnittstelle aktiviert.
2	Das Fenster wurde durch einen Mausklick des Anwenders aktiviert. Es kann jede Maustaste verwendet werden.

lParamHi: Ist ungleich 0, wenn das Fenster zum Symbol verkleinert wird. Andernfalls ist er 0.

lParamLo: Wenn *wParam* 0 ist, ein Handle des zu aktivierenden Fensters, andernfalls des zu deaktivierenden Fensters.

Rückgabewert Nicht verwendet.

Hinweis Wenn das Fenster aktiviert wird und nicht verkleinert ist, ordnet die Funktion *DefWindowProc* als Standardaktion dem Fenster den Fokus zu.

wm_ActivateAPP

Beschreibung Diese Botschaft wird übergeben, wenn das zu aktivierende Fenster einer anderen Anwendung angehört als das aktive Fenster. Die Botschaft wird sowohl an die Anwendung des zu aktivierenden Fensters, als auch an die Anwendung des deswegen zu deaktivierenden Fensters übergeben.

Parameter *wParam:* Bestimmt, ob das Fenster aktiviert oder deaktiviert werden soll. Der Wert 0 zeigt an, daß Windows ein Fenster deaktivieren wird. Andere Werte bedeuten, daß ein Fenster aktiviert wird.

lParam: Das Task-Handle der anderen Anwendung.

Rückgabewert Nicht verwendet.

wm_AskCBFormatName

Beschreibung Diese Botschaft wird übergeben, wenn die Zwischenablage ein Daten-Handle für das Format *cf_OwnerDisplay* enthält (d.h. der

gegenwärtige Besitzer der Zwischenablage sollte den Inhalt der Zwischenablage darstellen). Anschließend wird eine Kopie des Formatnamens angefordert.

Parameter *wParam:* Bestimmt die maximale Anzahl der zu kopierenden Bytes.

lParam: Zeigt auf den Puffer, der die Kopie des Formatnamens aufnehmen soll.

Rückgabewert Nicht verwendet.

Hinweis Der gegenwärtige Besitzer der Zwischenablage sollte beim Kopieren des Namens des Formats *cf_OwnerDisplay* die vorgegebene maximale Anzahl von Bytes nicht überschreiten. Format und Daten der Zwischenablage werden mit der Funktion *SetClipboardData* gesetzt.

wm_CancelMode

Beschreibung Diese Botschaft annuliert jeden Systemmodus, zum Beispiel den, der die Maus innerhalb einer Bildlaufleiste verfolgt oder der ein Fenster bewegt. Windows übergibt die Botschaft *wm_CancelMode*, sobald eine Anwendung ein Meldungsfenster darstellt.

Parameter *wParam:* Nicht verwendet.

lParam: Nicht verwendet.

Rückgabewert Nicht verwendet.

Hinweis Diese Botschaft teilt der Anwendung mit, daß Tastatur- und Mauseingabe auf das Meldungsfenster umgeleitet werden. Prozesse, die den Tastatur- bzw. Maustastenstatus und/oder die Mausposition überwachen, können nach dem Löschen des Meldungsfensters ungültig geworden sein.

wm_ChangeCBChain

Beschreibung Diese Botschaft teilt dem ersten Fenster in der Viewer-Kette der Zwischenablage mit, daß ein Fenster aus der Kette entfernt wird.

Parameter *wParam:* Enthält das Handle des aus der Viewer-Kette der Zwischenablage zu entfernenden Fensters.

lParamLo: Ist ein Handle des Fensters, das dem zu entfernenden Fenster in der Viewer-Kette der Zwischenablage nachfolgt.

lParamHi: Nicht verwendet.

Rückgabewert Nicht verwendet.

Hinweis

Jedes Fenster, das die Botschaft *wm_ChangeCBChain* empfängt, sollte die Funktion *SendMessage* aufrufen, um die Botschaft an das folgende Fenster in der Viewer-Kette der Zwischenablage weiterzugeben.

Ist das zu entfernende Fenster das in der Kette folgende Fenster, rückt das durch den Parameter *lParamLo* bezeichnete Fenster auf. Zwischenablagebotschaften werden an dieses Fenster weitergegeben. Fenster müssen sich selbst aus der Viewer-Kette der Zwischenablage entfernen, wenn sie die Botschaft *wm_Destroy* empfangen.

Weitere Informationen finden Sie in der Beschreibung der Botschaft *wm_DrawClipboard.*

wm_Char

Beschreibung

Diese Botschaft geht aus der Übersetzung einer *wm_KeyUp*- oder *wm_KeyDown*-Botschaft hervor. Sie enthält den Wert der gedrückten oder losgelassenen Taste.

Parameter

wParam: Enthält den Wert der Taste.

lParamLo: Wiederholungszähler. Der Wert ergibt sich aus der Zeit, die der Anwender die Taste gedrückt hält und entspricht der ermittelten Anzahl von Tastenwiederholungen.

lParamHi: Die Bits von *lParamHi* können folgende Werte annehmen:

Bit	Bedeutung
0 bis 7	Enthalten den Scan-Code der Taste (OEM-abhängiger Wert).
8	Ist 1, wenn die gedrückte Taste eine zusätzliche Taste (zum Beispiel Funktionstaste oder Taste des numerischen Tastenblocks) ist.
9 bis 12	Werden entweder überhaupt nicht oder nur intern von Windows verwendet.
13	Ist 1, wenn gleichzeitig mit der Taste die Alt-Taste gedrückt wurde.
14	Ist 1, wenn die Taste vor Übergabe dieser Botschaft bereits gedrückt war.
15	Ist 1, wenn die Taste losgelassen, 0, wenn sie gedrückt wurde.

Rückgabewert

Nicht verwendet.

Hinweis

Da es keine notwendige Eins-zu-Eins-Entsprechung zwischen gedrückten Tasten und erzeugten Zeichenbotschaften gibt, ist der Inhalt der Bits 8 bis 15 des Parameters *lParamHi* ohne Bedeutung für die Anwendung. Die Information dient aus-

schließlich der *wm_KeyUp*- oder *wm_KeyDown*-Botschaft, die der Übergabe der Zeichenbotschaft vorangeht.

Bei erweiterten IBM-Tastaturen mit 101 oder 102 Tasten gelten folgende Tasten als zusätzliche: die rechte Alt - und Ctrl - (bzw. Strg)-Taste im Hauptbereich der Tastatur, die Tasten Ins , Del , Home , End , PgUp , PgDn und die Cursor-Tasten im Block zwischen dem Nummernblock und der alphanumerischen Tastatur sowie die Divisionstaste / und die Eingabetaste im Nummernblock. Auch andere Tastaturen unterstützen möglicherweise das Zusatztasten-Bit im Parameter *lParamHi*. Nicht-System-Taste ist jede Taste, die gedrückt wird, wenn die Taste Alt nicht gedrückt wird.

Hat kein Fenster den Fokus, so werden die Botschaften *wm_SysKeyDown*, *wm_SysChar* und *wm_SysKeyUp* anstelle von *wm_KeyDown*, *wm_Char* und *wm_KeyUp* übergeben. Vergleichen Sie auch die Beschreibung der Botschaften *wm_DeadChar*, *wm_SysChar* und *wm_SysDeadChar*.

wm_CharToItem

Beschreibung Diese Botschaft wird von einer Liste des Typs *lbs_WantKeyboardInput* an ihren Besitzer als Reaktion auf eine *wm_Char*-Botschaft übergeben.

Parameter *wParam:* Enthält den Wert der vom Anwender gedrückten Taste.

lParamLo: Ein Handle zur Liste.

lParamHi: Die aktuelle Position des Caret.

Rückgabewert Der Rückgabewert bestimmt die Reaktion der Anwendung auf die Botschaft. Der Rückgabewert -2 zeigt an, daß die Anwendung die Auswahl eines Eintrags vollständig bearbeitet hat und keine weiteren Aktionen in der Liste erwartet werden. Der Rückgabewert -1 zeigt an, daß die Liste die dem Tastendruck entsprechende Aktion ausführen soll. Ein Rückgabewert von 0 oder größer bestimmt den Index des Listeneintrags und zeigt an, daß die Liste die dem Tastendruck und dem vorgegebenen Eintrag entsprechende Standardaktion ausführen soll.

Hinweis Vergleichen Sie dazu auch die Beschreibung der Botschaft *wm_VKeyToItem*.

wm_ChildActivate

Beschreibung Diese Botschaft wird an ein übergeordnetes Fenster übergeben, wenn die Funktion *SetWindowPos* das untergeordnete Fenster verschiebt.

Parameter	*wParam:* Nicht verwendet.
	lParam: Nicht verwendet.
Rückgabewert	Nicht verwendet.

wm_Clear

Beschreibung	Diese Botschaft löscht die aktuelle Auswahl in einem Fenster.
Parameter	*wParam:* Nicht verwendet.
	lParam: Nicht verwendet.
Rückgabewert	Nicht verwendet.

wm_Close

Beschreibung	Diese Botschaft tritt beim Schließen eines Fensters auf.
Parameter	*wParam:* Nicht verwendet.
	lParam: Nicht verwendet.
Rückgabewert	Nicht verwendet.
Hinweis	Die Funktion *DefWindowProc* ruft die Funktion *DestroyWindow* auf, um das Fenster zu schließen. Vor Abarbeitung der Botschaft *wm_Close* kann die Anwendung beim Anwender eine Bestätigung zur Schließung des Fensters abfragen. Stimmt der Anwender zu, wird die Funktion *DestroyWindow* aufgerufen.

wm_Command

Beschreibung	Diese Botschaft tritt auf, wenn der Anwender einen Menüeintrag auswählt, ein Steuerelement eine Botschaft an ihr übergeordnetes Fenster übergibt oder eine Accelerator-Taste übersetzt wird.
Parameter	*wParam:* Enthält den Menüeintrag, die ID eines Steuerelements oder die der Accelerator-Taste.
	lParamLo: Wenn *lParamLo* 0 ist, stammt die Botschaft von einem Menü, *lParamHi* wird in diesem Fall nicht verwendet. Ist *lParamHi* ungleich 0, hängt die Interpretation des Werts von *lParamHi* ab.
	lParamHi: Ist *lParamHi* 1 enthält *lParamLo* die ID der Accelerator-Taste. Andernfalls ist *lParamLo* das Handle eines Steuerelements und *lParamHi* ein Benachrichtigungscode, der an das übergeordnete Fenster des Steuerelements übergeben wird.
Rückgabewert	Nicht verwendet.

Hinweis Eine Accelerator-Taste ist für die Auswahl eines Eintrags aus dem Systemmenü definiert. Er wird in eine *wm_SysCommand*-Botschaften übersetzt.

Wird ein Accelerator-Tastendruck ausgeführt, der einem Menüeintrag entspricht, und ist das verwaltende Fenster des Menüs verkleinert, wird keine *wm_Command*-Botschaft übergeben.

Wenn aber der Accelerator-Tastendruck keinem Menüeintrag des Fensters oder des Sytemmenüs entspricht, wird eine *wm_Command*-Botschaft übergeben, auch wenn das betreffende Fenster verkleinert ist.

wm_Compacting

Beschreibung Diese Botschaft wird an alle Hauptfenster übergeben, sobald Windows feststellt, daß mehr als 12,5 Prozent der Systemzeit innerhalb eines 30 oder 60 Sekunden währenden Intervalls zur Neuanordnung des Speichers verwendet wird. In diesem Fall reicht der Sytemspeicher nicht aus.
Erhält eine Anwendung diese Botschaft, sollte sie möglichst viel Speicher bereitstellen und dabei die eigenen Anforderungen sowie die Anzahl der unter Windows laufenden Anwendungen berücksichtigen.

Deren Anzahl kann über die Funktion *GetNumTasks* ermittelt werden.

Parameter *wParam:* Bestimmt den Anteil der CPU-Zeit, die für die Speicherverwaltung aufgewendet wird. 32768 entspricht 50 % der CPU-Zeit.

lParam: Nicht verwendet.

Rückgabewert Nicht verwendet.

wm_CompareItem

Beschreibung Diese Botschaft bestimmt die relative Position eines neuen Eintrags in einem sortierten, besitzergezeichneten Kombinationsfenster oder einer Liste.

Fügt die Anwendung einen neuen Eintrag hinzu, übergibt Windows diese Botschaft an den Besitzer des mit *cbs_Sort* oder *lbs_Sort* erstellten Kombinationsfensters oder der Liste.

Der Parameter *lParam* ist ein Zeiger auf eine *TCompareItemStruct*-Datenstruktur, die Bezeichner und anwendungsunterstützte Daten zweier Einträge des Kombinationsfensters oder der Liste enthält.

Sobald der Besitzer die Botschaft erhält, gibt er einen Wert zurück, der die Reihenfolge der Einträge anzeigt.

Normalerweise übergibt Windows diese Botschaft mehrmals, bevor die genaue Position des neuen Eintrags bestimmt werden kann.

Parameter *wParam:* Nicht verwendet.

lParam: Enthält einen Zeiger auf eine *TCompareItemStruct*-Datenstruktur, die Bezeichner und anwendungsunterstützte Daten zweier Einträge in dem Kombinationsfenster oder der Liste enthält.

Rückgabewert Der Rückgabewert zeigt die relative Position zweier Einträge an. Folgende Werte sind möglich:

Wert	Bedeutung
-1	Eintrag 1 wird vor Eintrag 2 eingeordnet.
0	Eintrag 1 und 2 werden gleich eingeordnet.
1	Eintrag 1 wird nach Eintrag 2 eingeordnet.

wm_Copy

Beschreibung Diese Botschaft kopiert den aktuellen Auswahlbereich im *cf_Text*-Format in die Zwischenablage.

Parameter *wParam:* Nicht verwendet.

lParam: Nicht verwendet.

Rückgabewert Nicht verwendet.

wm_Create

Beschreibung Diese Botschaft teilt der Fensterprozedur mit, daß eine Initialisierung möglich ist. Sie geht von der Funktion *CreateWindow* aus und wird vor Ende dieser Funktion und vor dem Öffnen des Fensters übergeben.

Parameter *wParam:* Nicht verwendet.

lParam: Zeigt auf eine Datenstruktur *TCreateStruct*, in der eine Kopie der an die Funktion *CreateWindow* übergebenen Parameter abgelegt ist.

Rückgabewert Nicht verwendet.

wm_CtlColor

Beschreibung Diese Botschaft wird an das übergeordnete Fenster übergeben, wenn ein untergeordnetes Steuerelement oder ein Meldungsfenster gezeichnet werden soll. Dies ermöglicht es dem übergeordneten Fenster, Text- und Hintergrundfarben des unter-

geordneten Fensters unter Verwendung des in *wParam* vorgegebenen Handle des Bildschirmkontextes einzustellen.

Parameter *wParam:* Enthält das Handle des Bildschirmkontextes für das untergeordnete Fenster.

lParamLo: Ein Handle zum untergeordneten Fenster.

lParamHi: Eine der folgenden Konstanten, die den Typ des untergeordneten Fensters festlegen:

Wert	Elementtyp
ctlcolor_Btn	Schalter
ctlcolor_Dlg	Dialogfenster
ctlcolor_Edit	Editierfeld
ctlcolor_ListBox	Liste
ctlcolor_MsgBox	Meldungsfenster
ctlcolor_ScrollBar	Bildlaufleiste
ctlcolor_Static	Statisches Element

Rückgabewert Nicht verwendet.

Hinweis Die Funktion *DefWindowProc* wählt die Standardsystemfarben. Wird die Botschaft *wm_CtlColor* übergeben, muß die Anwendung den Ursprung des vorgesehenen Pinsels feststellen.

Das geschieht, indem zuerst die Funktion *UnrealizeObject* für den Pinsel aufgerufen wird. Anschließend kann der Ursprung des Pinsels auf die obere linke Ecke des Fensters gesetzt werden.

Bearbeitet eine Anwendung die Botschaft *wm_CtlColor*, so muß das Handle des für den Hintergrund vorgesehenen Pinsels zurückgegeben werden.

Die Übergabe eines ungültigen Handles kann einen Systemabsturz verursachen.

wm_Cut

Beschreibung Diese Botschaft übergibt die aktuelle Auswahl für die Zwischenablage im Format *cf_Text* und löscht die Auswahl anschließend aus dem Steuerelementfenster.

Parameter *wParam:* Nicht verwendet.

lParam: Nicht verwendet.

Rückgabewert Nicht verwendet.

wm_dde_Ack

Beschreibung Teilt einer Anwendung mit, daß eine andere DDE-Botschaft empfangen wurde.

Parameter *wParam*: Das Handle des die Botschaft sendenden Fensters.

lParam: War die empfangene Botschaft *wm_dde-Initiate*, enthält *lParamLo* ein Atom, das die antwortende Anwendung benennt. *lParamHi* enthält dann ein Atom mit dem Thema der Übertragung.

War die empfangene Botschaft *wm_dde_Execute*, enthält *lParamHo* einen Record, der den Status der Antwort bezeichnet. *lParamHi* ist dann ein Handle auf den Dateneintrag, der die Befehlszeile enthält.

Bei allen anderen Botschaften ist *lParamLo* ein Status-Record (TDDEACK) und *lParamHi* ein Atom, das den Dateneintrag bestimmt, für den die Antwort übergeben wird.

Rückgabewert Nicht verwendet.

Hinweis Diese Botschaft muß mit der Funktion *SendMessage* übergeben werden. Der erste Parameter sollte dabei das Handle des die Botschaft empfangenden Fensters sein.

wm_dde_Advise

Beschreibung Wird von der Client-Anwendung übergeben und fordert von der Server-Anwendung eine Aktualisierung, wenn sich der Dateneintrag verändert.

Parameter *wParam*: Das Handle des sendenden Fensters.

lParamLo: Eine *TDDEAdvise*-Struktur, das die Art des Datenaustausches bestimmt.

lParamHi: Ein Atom, das den angeforderten Dateneintrag bestimmt.

Rückgabewert Nicht verwendet.

Hinweis Diese Botschaft muß mit der Funktion *PostMessage* übergeben werden. Der erste Parameter sollte das Handle des die Botschaft empfangenden Fensters sein.

wm_dde_Data

Beschreibung Wird von der Server-Anwendung übergeben, um den Wert eines Dateneintrags zu übertragen oder um der Client-Anwendung die Verfügbarkeit des Eintrags anzuzeigen.

Parameter *wParam*: Das Handle des sendenden Fensters.

lParamLo: Das Handle des globalen Speicherblocks, der die *TDDEData*-Struktur mit den Daten enthält. *lParamLo* ist 0, wenn die Botschaft nur zum Anzeigen einer Veränderung verwendet wird.

lParamHi: Ein Atom, das festlegt, welcher Dateneintrag übergeben wird.

Rückgabewert Nicht verwendet.

Hinweis Diese Botschaft muß mit der Funktion *PostMessage* übergeben werden.

Der erste Parameter sollte das Handle des die Botschaft empfangenden Fensters sein.

wm_dde_Execute

Beschreibung Wird von der Client-Anwendung gesendet, um eine Befehlsfolge zu übertragen, die von der Server-Anwendung bearbeitet werden soll.

Parameter *wParam*: Das Handle des sendenden Fensters.

lParamLo: Reserviert.

lParamHi: Ein Handle auf das globale Speicherobjekt, das die Befehle enthält.

Rückgabewert Nicht verwendet.

Hinweis Diese Botschaft muß mit der Funktion *PostMessage* übergeben werden.

Der erste Parameter sollte das Handle des die Botschaft empfangenden Fensters sein.

wm_dde_Initiate

Beschreibung Wird von der Client- oder der Server-Anwendung gesendet, um eine Übertragung zu initialisieren. Die antwortende Anwendung sollte eine *wm_dde_Ack*-Botschaft übergeben.

Parameter *wParam*: Das Handle des sendenden Fensters.

lParamLo: Ein Atom, das den Namen der Anwendung festlegt, mit der die Übertragung stattfinden soll. Wenn *lParamLo* 0 ist, kann die Übertragung mit jeder Anwendung durchgeführt werden.

lParamHi: Ein Atom, das das Thema der gewünschten Übertragung bestimmt. Ist *lParamHi* 0, kann sich die Übertragung auf jedes Thema beziehen.

Rückgabewert Nicht verwendet.

Hinweis Diese Botschaft muß mit der Funktion *SendMessage* übergeben werden.

Der erste Parameter sollte das Handle des die Botschaft empfangenden Fensters sein.

wm_dde_Poke

Beschreibung	Wird von der Client-Anwendung gesendet, um die Server-Anwendung zum Empfang unerwünschter Daten zu veranlassen. Die Server-Anwendung antwortet mit einer *wm_dde_Ack*-Botschaft.
Parameter	*wParam*: Das Handle des sendenden Fensters.
	lParamLo: Ein Handle auf eine *TDDEPoke*-Struktur.
	lParamHi: Ein Atom, das den Dateneintrag bezeichnet.
Rückgabewert	Nicht verwendet.
Hinweis	Diese Botschaft muß mit der Funktion *PostMessage* übergeben werden. Der erste Parameter sollte das Handle des die Botschaft empfangenden Fensters sein.

wm_dde_Request

Beschreibung	Wird von der Client-Anwendung übergeben, um den Wert eines bestimmten Dateneintrags abzufragen.
Parameter	*wParam*: Das Handle des sendenden Fensters.
	lParamLo: Die Nummer eines Formats der Zwischenablage. Eine Beschreibung der Formate finden Sie in Kapitel 1 »Windows-Stile und -Konstanten« im Abschnitt »cf_Xxxx«.
	lParamHi: Ein Atom, das den gewünschten Dateneintrag bezeichnet.
Rückgabewert	Nicht verwendet.
Hinweis	Diese Botschaft muß mit der Funktion *PostMessage* übergeben werden. Der erste Parameter sollte das Handle des die Botschaft empfangenden Fensters sein.

wm_dde_Terminate

Beschreibung	Wird von einer Anwendung übergeben, um die Übertragung zu beenden.
Parameter	*wParam*: Das Handle des sendenden Fensters.
	lParam: Reserviert.
Rückgabewert	Nicht verwendet.
Hinweis	Diese Botschaft muß mit der Funktion *PostMessage* übergeben werden. Der erste Parameter sollte das Handle des die Botschaft empfangenden Fensters sein.

wm_dde_Unadvise

Beschreibung Wird von der Client-Anwendung an die Server-Anwendung übergeben, um diese zu informieren, daß die Aktualisierung eines bestimmten Dateneintrags oder des Zwischenablageformats dieses Eintrags nicht länger nötig ist.

Parameter *wParam*: Das Handle des sendenden Fensters.

lParamLo: Die Nummer eines Formats der Zwischenablage. Eine Beschreibung der Formate finden Sie in Kapitel 1 »Windows-Stile und -Konstanten« im Abschnitt »cf_Xxxx«.

lParamHi: Ein Atom, das den Dateneintrag bezeichnet.

Rückgabewert Nicht verwendet.

Hinweis Diese Botschaft muß mit der Funktion *PostMessage* übergeben werden. Der erste Parameter sollte das Handle des die Botschaft empfangenden Fensters sein.

wm_DeadChar

Beschreibung Diese Botschaft ergibt sich aus der Übersetzung einer *wm_KeyUp*- und *wm_KeyDown*-Botschaft. Sie bestimmt den Wert einer toten Taste.

Parameter *wParam:* Enthält den Wert der toten Taste.

lParamLo: Wiederholungszähler. Der Wert ergibt sich aus der Zeit, die der Anwender die Taste gedrückt hält und entspricht der ermittelten Anzahl von Tastenwiederholungen.

lParamHi: Die Bits von *lParamHi* können folgende Werte annehmen:

Bit	Bedeutung
0 bis 7	Enthalten den Scan-Code der Taste (OEM-abhängiger Wert).
8	Ist 1, wenn die gedrückte Taste eine zusätzliche Taste (zum Beispiel Funktionstaste oder Taste des numerischen Tastenblocks) ist.
9 bis 12	Werden entweder überhaupt nicht oder nur intern von Windows verwendet.
13	Ist 1, wenn gleichzeitig mit der Taste die Alt -Taste gedrückt wurde.
14	Ist 1, wenn die Taste vor Übergabe dieser Botschaft bereits gedrückt war.
15	Ist 1, wenn die Taste losgelassen, 0, wenn sie gedrückt wurde.

Rückgabewert Nicht verwendet.

Hinweis Die Botschaft *wm_DeadChar* kann von der Anwendung verwendet werden, um dem Anwender die Kontrolle über die tatsächlich betätigten Tasten zu ermöglichen. So kann ein Akzent an der aktuellen Caret-Position dargestellt werden, ohne daß das Caret bewegt werden müßte.

Da es keine notwendige Eins-zu-Eins-Entsprechung zwischen gedrückten Tasten und erzeugten Zeichenbotschaften gibt, ist der Inhalt der Bits 8 bis 15 des Parameters *lParamHi* ohne Bedeutung für die Anwendung. Die Information dient ausschließlich der *wm_KeyUp*- oder *wm_KeyDown*-Botschaft, die der Übergabe der Zeichenbotschaften vorangeht.

Bei erweiterten IBM-Tastaturen mit 101 oder 102 Tasten gelten folgende Tasten als zusätzliche: die rechte Alt - und Ctrl - (bzw. Strg)-Taste im Hauptbereich der Tastatur, die Tasten Ins , Del , Home , End , PgUp , PgDn und die Cursor-Tasten im Block zwischen der numerischen und der alphanumerischen Tastatur sowie die Divisions- und die Eingabetaste im numerischen Tastenblock. Auch andere Tastaturen unterstützen eventuell das Zusatztasten-Bit in*lParam*.

wm_DeleteItem

Beschreibung Diese Botschaft informiert den Besitzer einer besitzergezeichneten Liste oder eines Kombinationsfensters über die Entfernung eines Listeneintrags. Die Botschaft wird übergeben, wenn eine Liste oder ein Kombinationsfenster oder ein Eintrag aus ihnen durch eine der Botschaften *lb_DeleteString*, *lb_ResetContent*, *cb_DeleteString* oder *cb_ResetContent* entfernt wurde.

Parameter *wParam:* Nicht verwendet.

lParam: Enthält einen Zeiger auf eine *TDeleteItemStruct*-Datenstruktur, die Informationen über einen gelöschten Listeneintrag beinhaltet.

Rückgabewert Nicht verwendet.

wm_Destroy

Beschreibung Diese Botschaft meldet einem Fenster, daß es entfernt wird. Die Funktion *DestroyWindow* übergibt die Botschaft *wm_Destroy* nach dessen Entfernung vom Bildschirm an das übergeordnete Fenster. Sie wird außerdem vor der Entfernung eines Fensters an das übergeordnete Fenster übergeben.

Parameter *wParam:* Nicht verwendet.

lParam: Nicht verwendet.

Rückgabewert Nicht verwendet.

Hinweis Ist das zu löschende Fenster Teil der Viewer-Kette der Zwischenablage, die durch die Funktion *SetClipboardViewer* gesetzt wurde, muß das Fenster sich aus dieser Kette durch die Funktion *ChangeClipboardChain* vor der Rückkehr von der Botschaft *wm_Destroy* entfernen.

wm_DestroyClipboard

Beschreibung Diese Botschaft wird an den Besitzer der Zwischenablage übergeben, wenn diese durch einen Aufruf der Funktion *EmptyClipboard* geleert wird.

Parameter *wParam:* Nicht verwendet.

lParam: Nicht verwendet.

Rückgabewert Nicht verwendet.

wm_DevModeChange

Beschreibung Diese Botschaft wird nach Änderung einer Voreinstellung des Gerätemodus durch den Anwender an alle Hauptfenster übergeben.

Parameter *wParam:* Nicht verwendet.

lParam: Zeigt auf den in der Initialisierungsdatei von Windows, WIN.INI, festgelegten Gerätenamen.

Rückgabewert Nicht verwendet.

wm_DrawClipboard

Beschreibung Diese Botschaft wird an das erste Fenster in der Viewer-Kette der Zwischenablage übergeben, sobald sich der Inhalt der Zwischenablage verändert. Eine Bearbeitung dieser Botschaft ist nur dann nötig, wenn die Anwendung durch einen Aufruf der Funktion *SetClipboardViewer* in die Viewer-Kette der Zwischenablage eingefügt wurde.

Parameter *wParam:* Nicht verwendet.

lParam: Nicht verwendet.

Rückgabewert Nicht verwendet.

Hinweis Jedes Fenster, das die Botschaft *wm_DrawClipboard* erhält, sollte die Funktion *SendMessage* aufrufen, um sie an das nächste Fenster in der Viewer-Kette der Zwischenablage weiterzugeben. Das Handle des folgenden Fensters wird durch die

Funktion *SetClipboardViewer* zurückgegeben, Änderungen können sich jedoch aus einer zwischenzeitlichen *wm_ChangeCBChain*-Botschaft ergeben.

wm_DrawItem

Beschreibung Diese Botschaft teilt einem besitzergezeichneten Aktionsschalter, einem Kombinationsfenster, einer Liste oder einem Menü mit, daß ein sichtbarer Bestandteil des Steuerelementes verändert wurde. Das Feld *itemAction* in der *TDrawItemStruct*-Struktur bestimmt die durchzuführende Darstellungsoperation. Diese Daten erlauben es dem Besitzer der Steuerelemente, die Darstellungsoperation festzulegen.

Parameter *wParam:* Nicht verwendet.

lParam: Enthält einen Zeiger auf eine *TDrawItemStruct*-Datenstruktur, die Informationen über Gegenstand und Typ der Darstellungsoperation enthält.

Rückgabewert Nicht verwendet.

Hinweis Vor Abschluß der Bearbeitung dieser Botschaft sollte die Anwendung alle Objekte des Gerätekontextes, die vom Feld *hDC* der *TDrawItemStruct*-Datenstruktur unterstützt werden, wiederherstellen.

wm_Enable

Beschreibung Diese Botschaft wird nach Aktivierung oder Deaktivierung eines Fensters übergeben.

Parameter *wParam:* Bestimmt, ob das Fenster aktiviert oder deaktiviert wurde. Der Parameter *wParam* ist ungleich 0, wenn das Fenster aktiviert wurde, andernfalls hat er den Wert 0.

lParam: Nicht verwendet.

Rückgabewert Nicht verwendet.

wm_EndSession

Beschreibung Diese Botschaft wird an eine Anwendung übergeben, die auf die Botschaft *wm_QueryEndSession* einen Wert ungleich 0 zurückgegeben hat, um ihr mitzuteilen, daß die Arbeitssitzung beendet wird.

Parameter *wParam:* Bestimmt, ob die Sitzung beendet wird. In diesem Fall ist der Wert ungleich 0, andernfalls 0.

lParam: Nicht verwendet.

Rückgabewert Nicht verwendet.

Hinweis Hat der Parameter *wParam* einen Wert ungleich 0, kann Windows die Arbeitssitzung jederzeit nach Bearbeitung dieser Botschaft durch die Anwendungen beenden. Eine Anwendung sollte deshalb alle für die Beendigung notwendigen Aufgaben durchführen, bevor sie von der Bearbeitung dieser Botschaft zurückkehrt.

Ein Aufruf der Funktionen *DestroyWindow* oder *PostQuitMessage* ist nicht nötig.

wm_EnterIdle

Beschreibung Diese Botschaft teilt der Hauptfensterfunktion einer Anwendung mit, daß ein modales Dialogfenster oder ein Menü keine Aufgaben mehr haben. Dieser Status wird erreicht, wenn sich nach der Bearbeitung einer oder mehrerer Botschaften keine weiteren zu bearbeitenden Botschaften in der Warteschlange des modalen Dialogfensters bzw. des Menüs befinden.

Parameter *wParam:* Legt fest, ob die Botschaft Resultat eines dargestellten Dialogfensters oder Menüs ist. Der Parameter hat einen der folgenden Werte:

Wert	Bedeutung
msgf_DialogBox	Das System liegt wegen der Darstellung eines Dialogfensters still.
msgf_Menu	Das System liegt wegen der Darstellung eines Menüs still.

lParamLo: Ein Handle zum Dialogfenster oder Menü, wenn *wParam* den Wert *msgf_DialogBox* oder *msgf_Menu* hat.

lParamHi: Nicht verwendet.

Rückgabewert Nicht verwendet.

Hinweis Das System liegt still, weil ein modales Dialogfenster oder ein dargestelltes Menü keine Botschaften in der Warteschlange hat, nachdem mindestens eine vorhergehende Botschaft bearbeitet wurde. Der Standardrückgabewert der Funktion *DefWindowProc* ist 0.

wm_EraseBkgnd

Beschreibung Diese Botschaft wird übergeben, wenn der Hintergrund eines Fensters gelöscht werden muß, weil z.B. die Größe eines Fensters verändert wird. Mit ihrer Hilfe wird ein ungültig gewordener Bereich für das Zeichnen vorbereitet.

Parameter *wParam:* Enthält des Handle des Gerätekontextes.

lParam: Wird nicht verwendet.

Rückgabewert Bearbeitet eine Anwendung die Botschaft und löscht sie den Hintergrund des Fensters, sollte sie einen Wert ungleich 0 zurückgeben. Andernfalls sollte der Rückgabewert 0 sein.

Hinweis Die Standardaktion der Funktion *DefWindowProc* ist das Löschen des Hintergrundes unter Verwendung des Hintergrundpinsels der Klassenstruktur. Ist der Hintergrundpinsel der Klasse 0, sollte die Anwendung den Ursprung des Pinsels ausrichten, der für das Fenster verwendet werden soll.

Sie sollte die Funktion *UnrealizeObject* aufrufen, einen Pinsel auswählen und dann den Hintergrund unter Verwendung dieses Pinsels löschen. Windows geht dabei vom Abbildungsmoduls *mm_Text* aus.

Arbeitet der Bildschirmkontext mit einem anderen Abbildungsmodus, geht der gelöschte Bereich möglicherweise über den sichtbaren Bereich des Client-Bereiches hinaus.

wm_FontChange

Beschreibung Diese Botschaft tritt auf, wenn sich der Bestand der Schrift-Ressourcen verändert. Jede Anwendung, die Schriften zum System hinzufügt oder aus ihm entfernt (z.B. mit Hilfe der Funktionen *AddFontResource* oder *RemoveFontResource*), sollte diese Botschaft an alle Hauptfenster übergeben.

Parameter *wParam:* Wird nicht verwendet.

lParam: Wird nicht verwendet.

Rückgabewert Wird nicht verwendet.

Hinweis Um die Botschaft *wm_FontChange* an alle Hauptfenster zu übergeben, kann die Anwendung die Funktion *SendMessage* aufrufen.

wm_GetDlgCode

Beschreibung Diese Botschaft wird von Windows an die zu einem Steuerelement gehörende Eingabeprozedur übergeben. Üblicherweise gibt Windows alle Eingaben der Cursor-Tasten oder Tab-Taste an das Steuerelement ab. Eine Anwendung kann die Kontrolle einer bestimmten Art der Eingabe sowie deren Bearbeitung durch Beantworten der Botschaft *wm_GetDlgCode* übernehmen.

Parameter *wParam:* Wird nicht verwendet.

lParam: Wird nicht verwendet.

Rückgabewert Der Rückgabewert besteht in mindestens einem der folgenden Werte, die den Typ der durch die Anwendung überwachten Eingabe festlegen:

Wert	Bedeutung
dlgc_DefPushButton	Standardaktionsschalter.
dlgc_HasSetSel	*em_SetSel*-Botschaften.
dlgc_Button	Aktionsschalter.
dlgc_RadioButton	Schaltfeld.
dlgc_WantAllKeys	Alle Tastatureingaben.
dlgc_WantArrows	Cursor-Tasten.
dlgc_WantChars	*wm_Char*-Botschaften.
dlgc_WantMessage	Alle Tastatureingaben (die Anwendung leitet die Botschaft an das Steuerelement weiter).
dlgc_WantTab	Tab -Taste.

Hinweis Der Standardrückgabewert von *DefWindowProc* ist 0. Obwohl die Funktion *DefWindowProc* als Reaktion auf die Botschaft *wm_GetDlgCode* immer 0 zurückgibt, stellt der Rückgabewert der Fensterfunktionen der vordefinierten Steuerelementklassen einen der jeweiligen Klasse entsprechenden Code dar.

Die *wm_GetDlgCode*-Botschaft und die Rückgabewerte sind nur bei der Arbeit mit anwenderdefinierten Dialogsteuerelementen und unterklassifizierten Standardsteuerelementen hilfreich.

wm_GetFont

Beschreibung Diese Botschaft ruft die aktuell zur Textdarstellung in einem Dialogfenster verwendete Schrift ab.

Parameter *wParam:* Wird nicht verwendet.

lParam: Wird nicht verwendet.

Rückgabewert Der Rückgabewert entspricht dem Handle der Schrift, mit dem das Steuerelement arbeitet. Er ist 0, wenn die Systemschrift verwendet wird.

wm_GetMinMaxInfo

Beschreibung Diese Botschaft wird übergeben, wenn Windows die Maximalgröße eines Fensters, seine minimale oder maximale Tracking-Größe, oder dessen Position im vergrößerten Zustand benötigt. Das Vollbild eines Fensters ist erreicht, wenn seine Grenzen soweit als möglich ausgedehnt worden sind.

Die maximale Tracking-Größe entspricht der größtmöglichen Fenstergröße, die durch Größeneinstellung mit Hilfe der

Fensterränder erreicht werden kann. Die minimale Tracking-Größe entspricht der kleinstmöglichen Fenstergröße, die mit Hilfe der Fensterränder erreicht werden kann.

Parameter *wParam:* Wird nicht verwendet.

lParam: Zeigt auf ein Array mit fünf *TPoint*-Strukturen:

Point	Beschreibung
rgpt[0]	Zur internen Verwendung durch Windows.
rgpt[1]	Die Maximalgröße, sie ist die Standardeinstellung. Die Breite ergibt sich aus der Formel (*sm_CXScreen* + (2 * *sm_CXFrame*)), die Höhe aus (*sm_CYScreen* + (2 * *sm_CYFrame*)).
rgpt[2]	Die Position der oberen linken Ecke des vergrößerten Fensters in Bildschirmkoordinaten. Der Vorgabewert für x ist *sm_CXFrame*, für y *sm_CYFrame*.
rgpt[3]	Die kleinstmögliche Tracking-Größe, Vorgabe ist die Größe des Symbols. Die Breite ist *sm_CXMinTrack*, die Höhe *sm_CYMinTrack*.
rgpt[4]	Die größtmögliche Tracking-Größe, deren Vorgabewert kleiner als die Bildschirmgröße ist. Die Breite ergibt sich aus der Formel (*sm_CXScreen* + (2 * *sm_CXFrame*)), die Höhe aus (*sm_CYScreen* + (2 * *sm_CYFrame*)).

Rückgabewert Die Elemente 1 bis 4 des Arrays, auf das *lParam* zeigt, können nach Belieben geändert werden.

Hinweis Das Array enthält vor der Übergabe der Botschaft *wm_GetMinMaxInfo* durch Windows die Standardwerte für jeden Punkt. Mit Hilfe dieser Botschaft können die Standardwerte durch die Anwendung verändert werden.

wm_GetText

Beschreibung Diese Botschaft wird verwendet, um den zu einem Fenster gehörenden Text zu kopieren. Bei Editierfeldern und Editierfeldern in Kombinationsfenstern entspricht der zu kopierende Text dem Inhalt der Felder. Bei Schaltern entspricht der Text dem Namen des Schalters. In Listen entspricht der Text dem ausgewählten Eintrag. In anderen Fenstern entspricht der Text der Überschrift.

Parameter *wParam:* Bestimmt die maximale Anzahl der zu kopierenden Bytes, einschließlich des abschließenden Zeichens.

lParam: Zeigt auf den Puffer, der den Text aufnehmen soll. Er muß ausreichend groß sein, um die in *wParam* bestimmte Zeichenzahl aufzunehmen.

Rückgabewert Der Rückgabewert entspricht der Anzahl der kopierten Bytes, einschließlich des Null-Zeichens. Er ist *lb_Err*, wenn kein Eintrag ausgewählt war. Hat das Kombinationsfenster kein Editierfeld, ist er *cb_Err*.

Hinweis Weitere Informationen finden Sie in den Beschreibungen der Botschaften *wm_GetTextLength* und *wm_SetText*.

wm_GetTextLength

Beschreibung Diese Botschaft wird zur Ermittlung der Länge eines zu einem Fenster gehörenden Textes verwendet. In diese Angabe in Bytes ist die abschließende 0 nicht eingeschlossen. Bei Editierfeldern und Editierfeldern in Kombinationsfenstern ist der Text der Inhalt des Steuerelements. In Listen entspricht der Text dem ausgewählten Eintrag. Bei Schaltern entspricht der Text dem Schalternamen. In anderen Fenstern entspricht der Text der Überschrift.

Parameter *wParam:* Wird nicht verwendet.

lParam: Wird nicht verwendet.

Rückgabewert Der Rückgabewert entspricht der Länge des gegebenen Textes, ohne Berücksichtigung des abschließenden Nullzeichens.

Hinweis Weitere Informationen finden Sie in der Beschreibung der Botschaft *wm_GetText*.

wm_HScroll

Beschreibung Diese Botschaft wird übergeben, wenn der Anwender die horizontale Bildlaufleiste anklickt.

Parameter *wParam:* Enthält einen Bildlaufleistencode, der die Bildlaufwünsche des Anwenders angibt. Folgende Werte sind möglich:

Wert	Bedeutung
sb_Bottom	Bildlauf nach unten rechts.
sb_EndScroll	Ende des Bildlaufs.
sb_LineDown	Bildlauf eine Zeile nach unten.
sb_LineUp	Bildlauf eine Zeile nach oben.
sb_PageDown	Bildlauf eine Seite nach unten.
sb_PageUp	Bildlauf eine Seite nach oben.
sb_ThumbPosition	Bildlauf zu einer absoluten Position. Die aktuelle Position wird in *lParamLo* bereitgestellt.
sb_ThumbTrack	Mauszug zur festgelegten Position. Die aktuelle Position wird in *lParamLo* bereitgestellt.
sb_Top	Bildlauf nach oben links.

Parameter *lParamLo:* Die aktuelle Position des Bildlaufsymbols.

lParamHi: Bestimmt das Handle des Steuerelements der Bildlaufleiste. *lParamHi* ist 0, wenn die Bildlaufleiste mit dem Stil *ws_HScroll* erstellt wurde. Wird die Botschaft infolge eines Klicks des Anwenders auf die Bildlaufleiste eines Pop-Up-Fensters übergeben, wird lParamHi nicht verwendet.

Rückgabewert Nicht verwendet.

Hinweis Die Botschaft *sb_ThumbTrack* wird üblicherweise von Anwendungen benutzt, die auf Mauszüge reagieren.

Führt eine Anwendung einen Bildlauf in einem Fenster durch, muß sie die Mauszugposition durch die Funktion *SetScrollPos* zurücksetzen.

wm_HScrollClipboard

Beschreibung Diese Botschaft wird übergeben, wenn die Zwischenablage ein Daten-Handle auf das Format *cf_OwnerDisplay* enthält (der Besitzer der Zwischenablage sollte die enthaltenen Daten darstellen) und ein Ereignis an der horizontalen Bildlaufleiste der Zwischenablage vorliegt.

Parameter *wParam:* Enthält ein Handle zum Fenster der Zwischenablage.

lParam: Enthält einen der folgenden Bildlaufleistencodes im niederwertigen Word:

Wert	Bedeutung
sb_Bottom	Bildlauf nach unten rechts.
sb_EndScroll	Ende des Bildlaufs.
sb_LineDown	Bildlauf eine Zeile nach unten.
sb_LineUp	Bildlauf eine Zeile nach oben.
sb_PageDown	Bildlauf eine Seite nach unten.
sb_PageUp	Bildlauf eine Seite nach oben.
sb_ThumbPosition	Bildlauf zu einer absoluten Position.
sb_Top	Bildlauf nach oben links.

Rückgabewert Nicht verwendet.

Hinweis Der Besitzer der Zwischenablage sollte die Funktion *InvalidateRect* verwenden, oder wie gewünscht neu darstellen. Die Bildlaufleistenposition sollte zurückgesetzt werden.

Folgende Meldungen werden übergeben, wenn das Datenformat der Zwschenablage *cf_OwnerDisplay* ist: *wm_AskCBFormatName, wm_HScrollClipboard, wm_PaintClipboard* und *wm_VScrollClipboard.*

Daten und Format der Zwischenablage werden mit der Funktion *SetClipboardData* gesetzt.

wm_IconEraseBkgnd

Beschreibung Diese Botschaft wird an ein verkleinertes Fenster (Symbol) übergeben, wenn der Hintergrund vor Zeichnung des Symbols ausgemalt werden muß. Das Fenster empfängt diese Botschaft nur, wenn ein Klassensymbol für dieses Fenster definiert wurde, andernfalls wird *wm_EraseBkgnd* übergeben.

Die Übergabe dieser Botschaft an die Funktion *DefWindowProc* ermöglicht es Windows, den Hintergrund des Symbols mit dem Hintergrundpinsel des übergeordneten Fensters auszufüllen.

Parameter *wParam:* Enthält das Handle des Gerätekontextes des Symbols.

lParam: Wird nicht verwendet.

Rückgabewert Wird nicht verwendet.

wm_InitDialog

Beschreibung Diese Botschaft wird unmittelbar vor der Darstellung eines Dialogfensters übergeben. Durch Abarbeitung dieser Botschaft kann die Anwendung beliebige Initialisierungen vor Sichtbarmachung des Dialogfensters durchführen.

Parameter *wParam:* Bezeichnet das erste Steuerelement des Dialogfensters, dem der Fokus zugeordnet werden kann. Dieses ist im allgemeinen das erste vom Typ *ws_TabStop*.

lParam: Ist der Wert, der als Parameter *InitParam* übergeben wird, wenn das Dialogfenster durch eine der folgenden Funktionen erzeugt wird: *CreateDialogIndirectParam, CreateDialogParam, DialogBoxIndirectParam* und *DialogBoxParam.* Wurde das Dialogfenster mit *CrateDialog, CreateDialogIndirect, DialogBox* oder *DialogBoxIndirect* erzeugt, ist *lParam* 0. In allen anderen Fällen wird *lParam* nicht verwendet.

Rückgabewert Wird der Fokus einem der Steuerelemente eines Dialogfensters zugeordnet, kann der Rückgabewert 0 sein. Andernfalls muß er ungleich 0 sein.

Hinweis Gibt die Anwendung in Reaktion auf die Botschaft *wm_InitDialog* einen Wert ungleich 0 zurück, setzt Windows den Fokus auf den durch das Handle des Parameters *wParam* bezeichneten Eintrag.

wm_InitMenu

Beschreibung Diese Botschaft ist die Aufforderung zur Initialisierung eines Menüs. Sie tritt auf, wenn der Anwender eine Menüleiste

anklickt oder eine Menütaste betätigt. Windows übergibt diese Botschaft vor Darstellung des Menüs. Dies ermöglicht es der Anwendung, den Status der Menüeinträge vorab einzustellen.

Parameter *wParam:* Enthält das Menü-Handle des zu initialisierenden Menüs.

lParam: Wird nicht verwendet.

Rückgabewert Wird nicht verwendet.

Hinweis Die *wm_InitMenu*-Botschaft wird nur beim ersten Zugriff auf ein Menü übergeben, es wird nur eine Botschaft für jeden Zugriff erzeugt. Das Bewegen der Maus über die Einträge mit gedrückter Maustaste erzeugt also keine neuen Botschaften.

Die Botschaft übergibt keine Informationen über die Menüeinträge.

wm_InitMenuPopup

Beschreibung Diese Botschaft wird unmittelbar vor Darstellung eines Pop-up-Menüs übergeben. Die Bearbeitung der Botschaft ermöglicht es der Anwendung, den Status der Pop-up-Menüeinträge vorab und ohne Auswirkung auf den gesamten Menüstatus zu verändern.

Parameter *wParam:* Enthält das Menü-Handle des Pop-up-Menüs.

lParamLo: Der Index des Pop-up-Menüs im Hauptmenü.

lParamHi: Ist ungleich 0, wenn das Pop-up-Menü das Systemmenü ist. Andernfalls ist *lParamHi* 0.

Rückgabewert Wird nicht verwendet.

wm_KeyDown

Beschreibung Diese Botschaft wird bei Betätigung einer Standardtaste übergeben.

Eine Standardtaste ist jede Taste, die nicht zusammen mit der Alt-Taste gedrückt wird, sowie jede Taste, die betätigt wird, während ein Fenster den Fokus besitzt.

Parameter *wParam:* Bestimmt den virtuellen Tastencode der vorgegebenen Taste.

lParamLo: Wiederholungszähler. Der Wert ergibt sich aus der Zeit, die der Anwender die Taste gedrückt hält und entspricht der ermittelten Anzahl von Tastenwiederholungen.

lParamHi: Die Bits von *lParamHi* können folgende Werte annehmen:

Bit	Bedeutung
0 bis 7	Enthalten den Scan-Code der Taste (OEM-abhängiger Wert).
8	Ist 1, wenn die gedrückte Taste eine zusätzliche Taste (zum Beispiel Funktionstaste oder Taste des numerischen Tastenblocks) ist.
9 bis 12	Werden entweder überhaupt nicht oder nur intern von Windows verwendet.
13	Ist 1, wenn gleichzeitig mit der Taste die Alt -Taste gedrückt wurde.
14	Ist 1, wenn die Taste vor Übergabe dieser Botschaft bereits gedrückt war.
15	Ist 1, wenn die Taste losgelassen, 0, wenn sie gedrückt wurde.

Rückgabewert Wird nicht verwendet.

Hinweis Aufgrund der automatischen Wiederholung können einer *wm_KeyUp*-Botschaft mehr als eine *wm_KeyDown*-Botschaft vorangehen. Das Bit 14 von *lParamHi* kann verwendet werden, um festzustellen, ob es sich um die erste oder eine wiederholte *wm_KeyDown*-Botschaft handelt.

Bei erweiterten IBM-Tastaturen mit 101 oder 102 Tasten gelten folgende Tasten als zusätzliche: die rechte Alt - und Ctrl - (bzw. Strg)-Taste im Hauptbereich der Tastatur, die Tasten Ins , Del , Home , End , PgUp , PgDn und die Cursor-Tasten im Block zwischen dem Nummernblock und der alphanumerischen Tastatur sowie die Divisions- / und die Eingabetaste im Nummernblock. Auch andere Tastaturen unterstützen möglicherweise das Zusatztasten-Bit im Parameter *lParamHi*. Nicht-System-Taste ist jede Taste, die gedrückt wird, wenn die Taste Alt nicht gedrückt wird.

Hat kein Fenster den Fokus, so werden die Botschaften *wm_SysKeyDown*, *wm_SysChar* und *wmSysKeyUp* anstelle von *wm_KeyDown*, *wm_Char* und *wm_KeyUp* übergeben. Vergleichen Sie auch die Beschreibung der Botschaften *wm_DeadChar*, *wm_SysChar* und *wm_SysDeadChar*.

wm_KeyUp

Beschreibung Diese Botschaft wird beim Loslassen einer Standardtaste übergeben. Eine Standardtaste ist jede Taste, die nicht zusammen mit der Alt -Taste gedrückt wird, sowie jede Taste, die betätigt wird, während ein Fenster den Fokus besitzt.

Parameter *wParam:* Bestimmt den virtuellen Tastencode der vorgegebenen Taste.

lParamLo: Wiederholungszähler. Der Wert ergibt sich aus der Zeit, die der Anwender die Taste gedrückt hält und entspricht der ermittelten Anzahl von Tastenwiederholungen.

lParamHi: Die Bits von *lParamHi* können folgende Werte annehmen:

Bit	Bedeutung
0 bis 7	Enthalten den Scan-Code der Taste (OEM-abhängiger Wert).
8	Ist 1, wenn die gedrückte Taste eine zusätzliche Taste (zum Beispiel Funktionstaste oder Taste des numerischen Tastenblocks) ist.
9 bis 12	Werden entweder überhaupt nicht oder nur intern von Windows verwendet.
13	Ist 1, wenn gleichzeitig mit der Taste die Alt-Taste gedrückt wurde.
14	Ist 1, wenn die Taste vor Übergabe dieser Botschaft bereits gedrückt war.
15	Ist 1, wenn die Taste losgelassen, 0, wenn sie gedrückt wurde.

Rückgabewert Wird nicht verwendet.

Hinweis Bei erweiterten IBM-Tastaturen mit 101 oder 102 Tasten gelten folgende Tasten als zusätzliche: die rechte Alt - und Ctrl -(bzw. Strg -)Taste im Hauptbereich der Tastatur, die Tasten Ins , Del , Home , End , PgUp , PgDown und die Cursor-Tasten im Block zwischen dem Nummernblock und der alphanumerischen Tastatur sowie die Divisions- und die Eingabetaste im Nummernblock.

Auch andere Tastaturen unterstützen möglicherweise das Zusatztasten-Bit im Parameter *lParamHi*.

wm_KillFocus

Beschreibung Diese Botschaft wird unmittelbar bevor ein Fenster den Fokus verliert, übergeben.

Parameter *wParam:* Enthält das Handle des Fensters, das den Fokus besitzt.

lParam: Wird nicht verwendet.

Rückgabewert Wird nicht verwendet.

Hinweis Stellt eine Anwendung ein Caret dar, sollte dieses jetzt entfernt und freigegeben werden. *wParam* kann auch den Wert 0 haben.

wm_LButtonDblClk

Beschreibung Diese Botschaft tritt auf, wenn der Anwender einen Doppelklick mit der linken Maustatste durchführt.

Parameter *wParam:* Enthält einen Wert, der anzeigt, welche virtuellen Tasten gedrückt werden.

Möglich ist jede Kombination der in der folgenden Tabelle aufgeführten Werte:

Wert	Bedeutung
mk_Control	Gesetzt, wenn die Ctrl -Taste gedrückt wird.
mk_LButton	Gesetzt, wenn die linke Maustaste gedrückt wird.
mk_MButton	Gesetzt, wenn die mittlere Maustaste gedrückt wird.
mk_RButton	Gesetzt, wenn die rechte Maustaste gedrückt wird.
mk_Shift	Gesetzt, wenn die Shift -Taste gedrückt wird.

lParamLo: Die x-Koordinate des Mauszeigers.

lParamHi: Die y-Koordinate des Mauszeigers.

Rückgabewert Wird nicht verwendet.

Hinweis Die Koordinaten sind relativ zur oberen linken Ecke des Fensters. Nur Fenster, deren Fensterklasse den Stil *cs_DblClks* hat, können Doppelklickbotschaften empfangen.

Windows erzeugt eine Doppelklickbotschaft, wenn eine Maustaste gedrückt, losgelassen und innerhalb der vom System vorgegebenen Doppelklickzeit ein zweites Mal betätigt wird. Ein Doppelklick erzeugt vier Botschaften: Drücken der Taste, Loslassen der Taste, Doppelklickbotschaft, erneutes Loslassen der Taste.

Vergleichen Sie auch die Beschreibungen der folgenden Botschaften: *wm_LButtonDown, wm_LButtonUp, wm_MButtonDblClk, wm_MButtonDown, wm_MButtonUp, wm_MouseMove, wm_NCLButtonDblClk, wm_NCLButtonDown, wm_NCLButtonUp, wm_NCMButtonDblClk, wm_NCMButtonDown, wm_NCMButtonUp, wm_NCMouseMove, wm_NCRButtonDblClk, wm_NCRButtonDown, wm_NCRButtonUp, wm_RButtonDblClk, wm_RButtonDown* und *wm_RButtonUp.*

wm_LButtonDown

Beschreibung Diese Botschaft tritt auf, wenn der Anwender die linke Maustaste drückt.

Parameter *wParam:* Enthält einen Wert, der den Status verschiedener virtueller Tasten anzeigt.

Möglich ist jede Kombination der in der folgenden Tabelle aufgeführten Werte:

Wert	Bedeutung
mk_Control	Gesetzt, wenn die Ctrl -Taste gedrückt wird.
mk_MButton	Gesetzt, wenn die mittlere Maustaste gedrückt wird.
mk_RButton	Gesetzt, wenn die rechte Maustaste gedrückt wird.
mk_Shift	Gesetzt, wenn die Shift -Taste gedrückt wird.

lParamLo: Die x-Koordinate des Mauszeigers.

lParamHi: Die y-Koordinate des Mauszeigers.

Rückgabewert Wird nicht verwendet.

Hinweis Die Koordinaten sind relativ zur oberen linken Ecke des Fensters.

Vergleichen Sie auch die Beschreibungen folgender Botschaften: *wm_ButtonDblClk, wm_LButtonUp, wm_MButtonDblClk, wm_MButtonDown, wm_MButtonUp, wm_MouseMove, wm_NCLButtonDblClk, wm_NCLButtonDown, wm_NCLButtonUp, wm_NCMButtonDblClk, wm_NCMButtonDown, wm_NCMButtonUp, wm_NCMouseMove, wm_NCRButtonDblClk, wm_NCRButtonDown, wm_NCRButtonUp, wm_RButtonDblClk, wm_RButtonDown* und *wm_RButtonUp*.

wm_LButtonUp

Beschreibung Diese Botschaft tritt auf, wenn der Anwender die linke Maustaste losläßt.

Parameter *wParam:* Enthält einen Wert, der den Status verschiedener virtueller Tasten anzeigt. Möglich ist jede Kombination der in der folgenden Tabelle aufgeführten Werte:

Wert	Bedeutung
mk_Control	Gesetzt, wenn die Ctrl -Taste gedrückt wird.
mk_MButton	Gesetzt, wenn die mittlere Maustaste gedrückt wird.
mk_RButton	Gesetzt, wenn die rechte Maustaste gedrückt wird.
mk_Shift	Gesetzt, wenn die Shift -Taste gedrückt wird.

lParamLo: Die x-Koordinate des Mauszeigers.

lParamHi: Die y-Koordinate des Mauszeigers.

Rückgabewert Wird nicht verwendet.

Hinweis Die Koordinaten sind relativ zur oberen linken Ecke des Fensters.

Vergleichen Sie auch die Beschreibungen folgender Botschaften: *wm_ButtonDblClk, wm_LButtonDown, wm_MButtonDblClk, wm_MButtonDown, wm_MButtonUp, wm_MouseMove,*

wm_NCLButtonDblClk, wm_NCLButtonDown, wm_NCLButtonUp, wm_NCMButtonDblClk, wm_NCMButtonDown, wm_NCMButtonUp, wm_NCMouseMove, wm_NCRButtonDblClk, wm_NCRButtonDown, wm_NCRButtonUp, wm_RButtonDblClk, wm_RButtonDown und *wm_RButtonUp*.

wm_MButtonDblClk

Beschreibung Diese Botschaft tritt auf, wenn der Anwender einen Doppelklick mit der mittleren Maustaste ausführt.

Parameter *wParam:* Enthält einen Wert, der den Status verschiedener virtueller Tasten anzeigt. Jede Kombination der folgenden Werte ist möglich:

Wert	Bedeutung
mk_Control	Gesetzt, wenn die Ctrl -Taste gedrückt wird.
mk_LButton	Gesetzt, wenn die linke Maustaste gedrückt wird.
mk_MButton	Gesetzt, wenn die mittlere Maustaste gedrückt wird.
mk_RButton	Gesetzt, wenn die rechte Maustaste gedrückt wird.
mk_Shift	Gesetzt, wenn die Shift -Taste gedrückt wird.

lParamLo: Die x-Koordinate des Mauszeigers.

lParamHi: Die y-Koordinate des Mauszeigers.

Rückgabewert Wird nicht verwendet.

Hinweis Die Koordinaten sind relativ zur oberen linken Ecke des Fensters. Nur Fenster, deren Fensterklasse den Stil *cs_DblClks* hat, können Doppelklickbotschaften empfangen.

Windows erzeugt eine Doppelklickbotschaft, wenn eine Maustaste gedrückt, losgelassen und dann innerhalb der vom System vorgegebenen Doppelklickzeit ein zweites Mal betätigt wird. Ein Doppelklick erzeugt vier Botschaften:

- Drücken der Taste,
- Loslassen der Taste,
- Doppelklickbotschaft und
- erneutes Loslassen der Taste.

Vergleichen Sie auch die Beschreibungen folgender Botschaften: *wm_LButtonDblClk, wm_LButtonDown, wm_LButtonUp, wm_MButtonDown, wm_MButtonUp, wm_MouseMove, wm_NCLButtonDblClk, wm_NCLButtonDown, wm_NCLButtonUp, wm_NCMButtonDblClk, wm_NCMButtonDown, wm_NCMButtonUp, wm_NCMouseMove, wm_NCRButtonDblClk, wm_NCRButtonDown, wm_NCRButtonUp, wm_RButtonDblClk, wm_RButtonDown* und *wm_RButtonUp*.

wm_MButtonDown

Beschreibung Diese Botschaft tritt auf, wenn der Anwender die mittlere Maustaste drückt.

Parameter *wParam:* Enthält einen Wert, der den Status verschiedener virtueller Tasten anzeigt.

Jede Kombination der in der folgenden Tabelle aufgeführten Werte ist möglich:

Wert	Bedeutung
mk_Control	Gesetzt, wenn die Ctrl -Taste gedrückt wird.
mk_LButton	Gesetzt, wenn die linke Maustaste gedrückt wird.
mk_RButton	Gesetzt, wenn die rechte Maustaste gedrückt wird.
mk_Shift	Gesetzt, wenn die Shift -Taste gedrückt wird.

lParamLo: Die x-Koordinate des Mauszeigers.

lParamHi: Die y-Koordinate des Mauszeigers.

Rückgabewert Wird nicht verwendet.

Hinweis Die Koordinaten sind relativ zur oberen linken Ecke des Fensters.

Vergleichen Sie auch die Beschreibungen folgender Botschaften: *wm_LButtonDblClk, wm_LButtonDown, wm_LButtonUp, wm_MButtonDblClk, wm_MButtonUp, wm_MouseMove, wm_NCLButtonDblClk, wm_NCLButtonDown, wm_NCLButtonUp, wm_NCMButtonDblClk, wm_NCMButtonDown, wm_NCMButtonUp, wm_NCMouseMove, wm_NCRButtonDblClk, wm_NCRButtonDown, wm_NCRButtonUp, wm_RButtonDblClk, wm_RButtonDown* und *wm_RButtonUp*.

wm_MButtonUp

Beschreibung Diese Botschaft tritt auf, wenn der Anwender die mittlere Maustaste losläßt.

Parameter *wParam:* Enthält einen Wert, der den Status verschiedener virtueller Tasten anzeigt. Jede Kombination der folgenden Werte ist möglich:

Wert	Bedeutung
mk_Control	Gesetzt, wenn die Ctrl -Taste gedrückt wird.
mk_LButton	Gesetzt, wenn die linke Maustaste gedrückt wird.
mk_RButton	Gesetzt, wenn die rechte Maustaste gedrückt wird.
mk_Shift	Gesetzt, wenn die Shift -Taste gedrückt wird.

lParamLo: Die x-Koordinate des Mauszeigers.

lParamHi: Die y-Koordinate des Mauszeigers.

Rückgabewert Wird nicht verwendet.

Hinweis Die Koordinaten sind relativ zur oberen linken Ecke des Fensters.

Vergleichen Sie auch die Beschreibungen folgender Botschaften: *wm_LButtonDblClk, wm_LButtonDown, wm_LButtonUp, wm_MButtonDblClk, wm_MButtonDown, wm_MouseMove, wm_NCLButtonDblClk, wm_NCLButtonDown, wm_NCLButtonUp, wm_NCMButtonDblClk*, *wm_NCMButtonDown, wm_NCMButtonUp, wm_NCMouseMove, wm_NCRButtonDblClk, wm_NCRButtonDown, wm_NCRButtonUp, wm_RButtonDblClk, wm_RButtonDown* und *wm_RButtonUp.*

wm_MDIActivate

Beschreibung Eine Anwendung übergibt diese Botschaft an ein MDI-Client-Fenster (Multidokumentschnittstelle), um die Aktivierung eines untergeordneten MDI-Fensters zu veranlassen. Bei der Bearbeitung dieser Botschaft übergibt das Client-Fenster die Botschaft *wm_MDIActivate* an das zu aktivierende und an das zu deaktivierende untergeordnete Fenster.

Parameter *wParam:* Wenn die Anwendung die Botschaft *wm_MDIActivate* an das MDI-Client-Fenster übergibt, enthält der Parameter *wParam* das Fenster-Handle des zu aktivierenden untergeordneten MDI-Fensters. Übergibt das Client-Fenster die Botschaft an ein untergeordnetes Fenster, so hat der Parameter *wParam* einen Wert ungleich 0, wenn es aktiviert, 0, wenn es deaktiviert wird.

lParamLo: Das Handle des zu aktivierenden untergeordneten MDI-Fensters.

lParamHi: Das Handle des zu deaktivierenden untergeordneten MDI-Fensters.

Rückgabewert Wird nicht vergeben.

Hinweis Untergeordnete MDI-Fenster werden unabhängig vom MDI-Rahmenfenster aktiviert. Wird der Rahmen aktiv, erhält das zuletzt mit der Botschaft *wm_MDIActivate* aktivierte, untergeordnete Fenster die Botschaft *wm_NCActivate*, um Rahmen und Fenstertitel zu zeichnen. Weitere *wm_MDIActivate*-Botschaften werden nicht an das Fenster übergeben.

Falls das zu deaktivierende untergeordnete MDI-Fenster auf maximale Größe eingestellt ist, wird seine vorherige Größe wiederhergestellt und das zu aktivierende untergeordnete MDI-Fenster wird auf maximale Größe eingestellt.

wm_MDICascade

Beschreibung Diese Botschaft überführt die einem MDI-Client-Fenster (Multidokumentschnittstelle) untergeordneten Fenster in das Format *Cascade* (überlappend).

Parameter *wParam:* Wird nicht verwendet.

lParam: Wird nicht verwendet.

Rückgabewert Wird nicht verwendet.

wm_MDICreate

Beschreibung Diese Botschaft veranlaßt ein MDI-Client-Fenster (Multidokumentschnittstelle) zur Erstellung eines untergeordneten Fensters.

Parameter *wParam:* Wird nicht verwendet.

lParam: Enthält einen Zeiger auf eine Datenstruktur *TMDICreateStruct*.

Rückgabewert Der Rückgabewert enthält den Bezeichner des neuen Fensters im niederwertigen Word und 0 im höherwertigen Word.

Hinweis Das Fenster wird mit den Stil-Bits *ws_Child, ws_ClipSiblings, ws_ClipChildren, ws_SysMenu, ws_Caption, ws_ThickFrame, ws_MinimizeBox* und *ws_MaximizeBox*, sowie den zusätzlichen Stil-Bits in der durch *lParam* bezeichneten *TMDICreateStruct*-Datenstruktur erstellt. Windows fügt die Überschrift des neuen Fensters in das Fenstermenü des Rahmenfensters ein. Die Anwendung sollte alle dem Client-Fenster untergeordneten Fenster mit dieser Botschaft erzeugen.

Empfängt ein Client-Fenster eine Botschaft, welche die Aktivierung untergeordneter Fenster beeinflußt und ist das aktive Fenster vergrößert, stellt Windows das bisher aktivierte Fenster in alter Größe wieder her und vergrößert das neu aktivierte Fenster.

Bei Erzeugung des untergeordneten MDI-Fensters übergibt Windows die Botschaft *wm_Create* an das Fenster. Der Parameter *lParam* dieser Botschaft enthält einen Zeiger auf eine *TCreateStruct*-Datenstruktur. Das Feld *lpCreateParams* dieser Datenstruktur enthält einen Zeiger auf die *TMDICreateStruct*-Datenstruktur, die mit der das untergeordnete MDI-Fenster erzeugenden Botschaft *wm_MDICreate* übergeben wird.

Eine Anwendung sollte keine zweite *wm_MDICreate*-Botschaft übergeben, während die vorhergehende noch bearbeitet wird. Es sollte zum Beispiel keine *wm_MDICreate*-Botschaft übergeben, während ein untergeordnetes MDI_Fenster seine *wm_Create*-Botschaft abarbeitet.

wm_MDIDestroy

Beschreibung	Empfängt ein MDI-Client-Fenster (Multidokumentschnittstelle) diese Botschaft, wird ein untergeordnetes Fenster geschlossen.
Parameter	*wParam:* Enthält das Fenster-Handle des untergeordneten Fensters. *lParam:* Wird nicht verwendet.
Rückgabewert	Wird nicht verwendet.
Hinweis	Diese Botschaft löscht die Überschrift des untergeordneten Fensters aus dem Rahmenfenster und schließt es dann. Anwendungen sollten alle untergeordneten MDI-Fenster mit Hilfe dieser Botschaft schließen. Erhält ein Client-Fenster eine Botschaft, welche den Aktivierungsstatus eines untergeordneten Fensters verändert und ist gleichzeitig das derzeit aktive untergeordnete MDI-Fenster vergrößert, wird es durch Windows auf normale Größe gebracht. Außerdem wird das neu aktivierte untergeordnete Fenster vergrößert.

wm_MDIGetActive

Beschreibung	Diese Botschaft liefert das aktive untergeordnete MDI-Fenster (Multidokumentschnittstelle) zusammen mit einem Flag, das anzeigt, ob das Fenster vergrößert ist.
Parameter	*wParam:* Wird nicht verwendet. *lParam*: Wird nicht verwendet.
Rückgabewert	Der Rückgabewert enthält im niederwertigen Word das Handle des untergeordneten MDI-Fensters. Das höherwertige Word enthält den Wert 1, wenn das Fenster vergrößert ist; andernfalls ist der Wert 0.

wm_MDIIconArrange

Beschreibung	Diese Botschaft wird an ein MDI-Client-Fenster (Multidokumentschnittstelle) übergeben, um die verkleinerten untergeordneten Dokumentenfenster anzuordnen. Es werden nur verkleinerte Fenster (Symbole) angeordnet.
Parameter	*wParam:* Wird nicht verwendet. *lParam:* Wird nicht verwendet.
Rückgabewert	Wird nicht verwendet.

wm_MDIMaximize

Beschreibung	Diese Botschaft veranlaßt ein MDI-Client-Fenster (Multidokumentschnittstelle), ein untergeordnetes MDI-Fenster zu vergrößern. Windows paßt die Größe des zu öffnenden Fensters an das Client-Fenster an, wenn es zu groß ist. Das Systemmenü des untergeordneten Fensters wird in der Menüzeile des Rahmens positioniert und die Überschrift der Überschrift des Rahmenfensters hinzugefügt. Der Anwender kann so jederzeit das untergeordnete Fenster schließen oder wiederherstellen.
Parameter	*wParam:* Enthält den Bezeichner des untergeordneten Fensters. *lParam:* Wird nicht verwendet.
Rückgabewert	Wird nicht verwendet.
Hinweis	Erhält ein Client-Fenster eine Botschaft, welche den Aktivierungsstatus eines untergeordneten Fensters verändert und ist gleichzeitig das derzeit aktive untergeordnete MDI-Fenster vergrößert, wird es durch Windows wiederhergestellt. Außerdem wird das neu aktivierte untergeordnete Fenster vergrößert.

wm_MDINext

Beschreibung	Aktiviert das dem aktiven unmittelbar folgende, untergeordnete MDI-Fenster (Multidokumentschnittstelle) und positioniert das noch aktive hinter allen anderen untergeordneten Fenstern.
Parameter	*wParam:* Wird nicht verwendet. *lParam:* Wird nicht verwendet.
Rückgabewert	Wird nicht verwendet.
Hinweis	Erhält ein Client-Fenster eine Botschaft, welche den Aktivierungsstatus eines untergeordneten Fensters verändert und ist gleichzeitig das derzeit aktive untergeordnete MDI-Fenster vergrößert, wird es durch Windows wiederhergestellt. Außerdem wird das neu aktivierte untergeordnete Fenster vergrößert.

wm_MDIRestore

Beschreibung	Diese Botschaft stellt ein untergeordnetes MDI-Fenster (Multidokumentschnittstelle) aus dem Vollbild oder der Symboldarstellung wieder her.
Parameter	*wParam:* Der Bezeichner des untergeordneten Fensters. *lParam:* Wird nicht verwendet.

Rückgabewert Wird nicht verwendet.

wm_MDISetMenu

Beschreibung Diese Botschaft tauscht das Menü eines MDI-Rahmenfensters (Multidokumentschnittstelle), das Pop-up-Menü des Fensters oder beide aus.

Parameter *wParam:* Wird nicht verwendet.

lParamLo: Das Handle eines neuen MDI-Rahmenfensters oder 0.

lParamHi: Das Handle eines neuen Pop-up_Menüs oder 0.

Rückgabewert Der Rückgabewert ist das Handle des Rahmenfenstermenüs, das durch die Botschaft ersetzt wird.

Hinweis Nach Übergabe dieser Botschaft muß die Anwendung die Funktion *DrawMenuBar* zur Aktualisierung der Menüleiste aufrufen. Tauscht die Botschaft ein Fenster-Pop-up-Menü aus, werden die Menüeinträge des untergeordneten MDI-Fensters aus dem vorhergehenden Fenstermenü gelöscht und zum neuen Pop-Up-Menü hinzugefügt. Tauscht diese Botschaft ein MDI-Rahmenfenstermenü aus, während ein untergeordnetes MDI-Fenster vergrößert dargestellt wird, werden das Systemmenü und die Wiederherstellungssteuerelemente aus dem vorhergehenden Rahmenfenstermenü entfernt und in das neue Menü eingefügt.

wm_MDITile

Beschreibung Diese Botschaft veranlaßt ein MDI-Client-Fenster (Multidokumentschnittstelle) alle ihm untergeordneten Fenster im Format *Tile* (nebeneinander) darzustellen.

Parameter *wParam:* Wird nicht verwendet.

lParam: Wird nicht verwendet.

Rückgabewert Wird nicht verwendet.

wm_MeasureItem

Beschreibung Diese Botschaft wird an den Besitzer eines besitzergezeichneten Schalters, Kombinationsfensters, Menüeintrages oder einer Liste übergeben, wenn das betreffende Element erzeugt wird.

Erhält der Besitzer diese Botschaft, füllt er vor der Antwort die durch den Parameter *lParam* bezeichnete *TMeasureItemStruct*-Datenstruktur aus.

Diese Datenstruktur informiert Windows über die Dimensionen des Steuerelementes. Werden die Typen *lbs_OwnerDrawVariable* oder *cbs_OwnerDrawVariable* zur Erstellung der Liste oder des Kombinationsfensters verwendet, empfängt der Besitzer eine Botschaft für jeden Eintrag im Steuerelement. In allen anderen Fällen wird die Botschaft nur einmal empfangen.

Parameter *wParam:* Wird nicht verwendet.

lParam: Enthält einen Zeiger auf eine *TMeasureItemStruct*-Datenstruktur. Diese beinhaltet die Dimensionen des gezeichneten Steuerelements.

Rückgabewert Wird nicht verwendet.

Hinweis Windows übergibt die Botschaft *wm_MeasureItem* vor Übergabe der *wm_InitDialog*-Botschaft an den Besitzer eines Kombinationsfensters oder einer Liste, wenn diese mit dem Stil-Typ *cbs_OwnerDrawFixed* oder *lbs_OwnerDrawFixed* erstellt wurden.

wm_MenuChar

Beschreibung Diese Botschaft wird übergeben, wenn der Anwender eine Tastenkombination betätigt, die keiner vordefinierten Tastenkombination des aktuellen Menüs entspricht. Die Botschaft wird an das Fenster übergeben, welches das Menü verwaltet.

Parameter *wParam:* Enthält den Wert des vom Anwender gedrückten ASCII-Zeichens.

lParamLo: mf_Popup, wenn das Menü ein Pop-up-Menü ist, *mf_SysMenu*, wenn es sich um ein Systemmenü handelt.

lParamHi: Das Handle des aktuellen Menüs.

Rückgabewert Das höherwertige Word des Rückgabewerts enthält einen der folgenden Befehlscodes:

Code	Bedeutung
0	Weist Windows an, das vom Benutzer gewählte Zeichen zu verwerfen und erzeugt ein Warnsignal im Systemlautsprecher.
1	Weist Windows an, das aktuelle Menü zu schließen.
2	Informiert Windows, daß im niederwertigen Word des Rückgabewerts die Eintragsnummer eines bestimmten Eintrags abgelegt ist. Der betreffende Eintrag wird von Windows ausgewählt.

Hinweis Das niederwertige Word wird ignoriert, wenn das höherwertige den Wert 0 oder 1 enthält.

Werden Accelerator-Tasten verwendet, um in einem Menü positionierte Bitmaps auszuwählen, sollte die Anwendung diese Botschaft bearbeiten.

wm_MenuSelect

Beschreibung Die Botschaft tritt nach Auswahl eines Menüeintrags durch den Anwender auf.

Parameter *wParam:* Bezeichnet den ausgewählten Eintrag. Handelt es sich um einen Menüeintrag, enthält der Parameter *wParam* die zugehörige ID. Enthält der ausgewählte Eintrag ein Pop-up-Menü, so enthält *wParam* dessen Handle.

lParamLo: Entweder -1, oder eine Kombination der folgenden Werte:

Wert	Bedeutung
mf_Bitmap	Der Eintrag ist ein Bitmap.
mf_Checked	Der Eintrag ist markiert.
mf_Disabled	Der Eintrag ist deaktiviert.
mf_Grayed	Der Eintrag ist als nicht aktivierbar markiert.
mf_MouseSelect	Der Eintrag wurde mit der Maus ausgewählt.
mf_OwnerDraw	Der Eintrag ist besitzergezeichnet.
mf_Popup	Der Eintrag enthält ein Pop-up-Menü.
mf_SysMenu	Der Eintrag ist im Systemmenü enthalten. Das höherwertige Word enthält das Menü, auf das sich die Botschaft bezieht.

Hinweis Enthält *lParamLo* den Wert -1 und *lParamHi* den Wert 0, so wurde das Menü von Windows geschlossen, weil der Anwender die Esc -Taste gedrückt oder außerhalb des Menüs geklickt hat.

wm_MouseActivate

Beschreibung Diese Botschaft tritt auf, wenn sich der Mauszeiger in einem nicht aktiven Fenster befindet und eine Maustaste gedrückt wird. Das übergeordnete Fenster erhält diese Botschaft nur, wenn sie vom untergeordneten Fenster an die Funktion *DefWindowProc* weitergegeben wird.

Parameter *wParam:* Enthält das Handle des höchsten, dem zu aktivierenden Fenster übergeordneten Fensters.

lParamLo: Eine der *ht*-Konstanten. Diese Konstanten sind im Abschnitt »ht_Xxxx« in Kapitel 1 »Windows-Stile und -Konstanten« beschrieben.

Diese Werte werden auch von der Botschaft *wm_NCHitTest* zurückgegeben.

lParamHi: Die Nummer der Mausmeldung.

Rückgabewert Ist der Rückgabewert *ma_Activate*, wird das erste Fenster aktiviert, das die Botschaft erhält. Ist der Rückgabewert *ma_NoActivate*, wird das erste Fenster, das die Botschaft erhält, nicht aktiviert. Ist der Rückgabewert *ma_ActivateAndEat*, wird das erste Fenster aktiviert, das die Botschaft erhält. Das Mausereignis wird verworfen. Andere Rückgabewerte sind nicht zulässig.

Hinweis Übergibt ein untergeordnetes Fenster die Botschaft an die Funktion *DefWindowProc*, wird sie von dieser Funktion vor jeder Bearbeitung an das übergeordnete Fenster weitergeleitet. Gibt das übergeordnete Fenster einen Wert ungleich 0 zurück, wird die Bearbeitung gestoppt.

wm_MouseMove

Beschreibung Diese Botschaft tritt auf, wenn der Anwender die Maus bewegt, während sich der Mauszeiger im Client-Bereich des Fensters befindet.

Parameter *wParam:* Enthält einen Wert, der den Zustand verschiedener virtueller Tasten anzeigt. Jede Kombination der folgenden Werte ist möglich:

Wert	Bedeutung
mk_Control	Gesetzt, wenn die Ctrl -Taste gedrückt ist.
mk_LButton	Gesetzt, wenn die linke Maustaste gedrückt ist.
mk_MButton	Gesetzt, wenn die mittlere Maustaste gedrückt ist.
mk_RButton	Gesetzt, wenn die rechte Maustaste gedrückt ist.
mk_Shift	Gesetzt, wenn die Shift -Taste gedrückt ist.

lParamHi: Die y-Koordinate des Mauszeigers.

Rückgabewert Wird nicht verwendet.

Hinweis Die Koordinaten sind relativ zur oberen linken Ecke des Fensters.

Vergleichen Sie auch die Beschreibungen folgender Botschaften: *wm_LButtonDblClk, wm_LButtonDown, wm_LButtonUp, wm_MButtonDblClk, wm_MButtonDown, wm_ButtonUp, wm_NCLButtonDblClk, wm_NCLButtonDown, wm_NCLButtonUp, wm_NCMButtonDblClk, wm_NCMButtonDown, wm_NCMButtonUp, wm_NCMouseMove, wm_NCRButtonDblClk, wm_NCRButtonDown, wm_NCRButtonUp, wm_RButtonDblClk, wm_RButtonDown* und *wm_RButtonUp*.

wm_Move

Beschreibung	Diese Botschaft wird nach dem Verschieben eines Fensters übergeben.
Parameter	*wParamLo:* Die neue x-Koordinate der oberen linken Ecke des Client-Bereichs des Fensters.
	lParamHi: Die neue y-Koordinate der oberen linken Ecke des Client-Bereichs des Fensters.
Rückgabewert	Wird nicht verwendet.
Hinweis	Die Koordinaten von Pop-up- und überlappenden Fenstern sind Bildschirmkoordinaten. Die Koordinaten von untergeordneten Fenstern sind relativ zum Client-Bereich des übergeordneten Fensters.

wm_NCActivate

Beschreibung	Diese Botschaft wird an ein Fenster übergeben, wenn dessen Nicht-Client-Bereich verändert werden muß, um den aktiven bzw. inaktiven Status anzuzeigen.
Parameter	*wParam:* Bestimmt, ob eine Titelleiste oder ein Symbol verändert werden muß, um den aktiven bzw. inaktiven Status anzuzeigen. Der Parameter *wParam* ist ungleich 0, wenn ein aktiver Titel bzw. ein aktives Symbol dargestellt werden soll. Bei inaktiven Titeln oder Symbolen ist er 0.
	lParam: Wird nicht verwendet.
Rückgabewert	Wird nicht verwendet.
Hinweis	Die Funktion *DefWindowProc* zeichnet eine graue Titelleiste für inaktive Fenster und eine schwarze Titelleiste für aktive Fenster.

wm_NCCalcSize

Beschreibung	Diese Botschaft wird übergeben, wenn die Größe des Client-Bereichs eines Fensters berechnet werden muß.
Parameter	*wParam:* Wird nicht verwendet.
	lParam: Zeigt auf eine *TRect*-Datenstruktur, in der die Koordinaten des Fensterrechtecks (einschließlich Client-Bereich, Grenzlinien, Überschrift, Bildlaufleisten usw.) abgelegt sind.
Rückgabewert	Wird nicht verwendet.
Hinweis	Die Funktion *DefWindowProc* berechnet die Größe des Client-Bereichs, ausgehend von den Daten des Fensters (Existenz von

Bildlaufleisten, Menüs, usw.). Das Ergebnis wird in der *TRect*-Datenstruktur abgelegt.

wm_NCCreate

Beschreibung: Diese Botschaft wird noch vor der Botschaft *wm_Create* beim Erzeugen eines Fensters übergeben.

Parameter: *wParam:* Enthält das Handle des zu erstellenden Fensters.

lParam: Zeigt auf die *TCreateStruct*-Datenstruktur des Fensters.

Rückgabewert: Der Rückgabewert ist ungleich 0, wenn der Nicht-Client-Bereich erzeugt wird. Wenn ein Fehler auftritt, wird 0 zurückgegeben.

Der Rückgabewert der Funktion *CreateWindow* ist in diesem Fall 0.

Hinweis: Bildlaufleisten werden initialisiert (Setzen von Position und Anzeigebereich der Bildlaufleiste) und der Fenstertext gesetzt. Intern verwendeter Speicher für Erzeugung und Verwaltung des Fensters wird reserviert. Diese Aufgaben übernimmt die Funktion *DefWindowProc*.

wm_NCDestroy

Beschreibung: Diese Botschaft teilt einem Fenster mit, daß sein Nicht-Client-Bereich freigegeben und entfernt wird. Die Funktion *DestroyWindow* übergibt die Botschaft *wm_NCDestroy* im Anschluß an die *wm_Destroy*-Botschaft an das Fenster. Diese Botschaft wird verwendet, um den dem Fenster zugewiesenen Speicherbereich freizugeben.

Parameter: *wParam:* Wird nicht verwendet.

lParam: Wird nicht verwendet.

Rückgabewert: Wird nicht verwendet.

wm_NCHitTest

Beschreibung: Diese Botschaft wird an das Fenster übergeben, das den Mauszeiger enthält oder das die Funktion *SetCapture* verwendet hat, um die Mauseingabe auf sich zu ziehen. Die Botschaft wird bei jeder Mausbewegung übergeben.

Parameter: *wParam:* Wird nicht verwendet.

lParamLo: Die x-Koordinate des Mauszeigers.

lParamHi: Die y-Koordinate des Mauszeigers.

Rückgabewert

Wird diese Botschaft an die Funktion *DefWindowProc* übergeben, dann ist der Rückgabewert eine der folgenden Konstanten:

Konstante	Bedeutung
htBottom	Auf der unteren horizontalen Grenzlinie des Fensters.
htBottomLeft	Auf der unteren linken Ecke der Fenstergrenze.
htBottomRight	Auf der unteren rechten Ecke der Fenstergrenze.
htCaption	In einer Überschrift.
htClient	In einem Client-Bereich.
htError	Wie *htNowhere*. Zusätzlich erzeugt die Funktion *DefWindowProc* einen Warnlaut, um einen Fehler anzuzeigen.
htGrowBox	In einem Größenveränderungsfeld.
htHScroll	In der horizontalen Bildlaufleiste.
htLeft	Auf der linken Grenzlinie des Fensters.
htMenu	In einem Menübereich.
htNowhere	Auf dem Bildschirmhintergrund oder einer Trennlinie zwischen zwei Fenstern.
htReduce	In einem Verkleinerungsfeld.
htRight	Auf der rechten Grenzlinie des Fensters.
htSize	Wie *htGrowBox*.
htSysMenu	In einem Menüfeld (Schließfeld in untergeordneten Fenstern).
htTop	Auf der oberen horizontalen Grenzlinie des Fensters.
htTopLeft	Auf der oberen linken Ecke der Fenstergrenze.
htTopRight	Auf der oberen rechten Ecke der Fenstergrenze.
htTransparent	In einem Fenster, auf dem ein anderes Fenster liegt.
htVScroll	In der vertikalen Bildlaufleiste.
htZoom	In einem Vergrößerungsfeld.

Hinweis

Die Koordinaten sind relativ zur oberen linken Ecke des Fensters.

Normalerweise erhält das Fenster alle Botschaften, in dem sich der Mauszeiger befindet. Dies kann mit der Funktion *SetCapture* umgangen werden.

Vergleichen Sie auch die Beschreibungen folgender Botschaften: *wm_LButtonDblClk*, *wm_LButtonDown*, *wm_LButtonUp*, *wm_MButtonDblClk*, *wm_MButtonDown*, *wm_MButtonUp*, *wm_MouseMove*, *wm_NCLButtonDblClk*, *wm_NCLButtonDown*, *wm_NCLButtonUp*, *wm_NCMButtonDblClk*, *wm_NCMButtonDown*, *wm_NCMButtonUp*, *wm_NCMouseMove*, *wm_NCRButtonDblClk*, *wm_NCRButtonDown*, *wm_NCRButtonUp*, *wm_RButtonDblClk*, *wm_RButtonDown* und *wm_RButtonUp*.

wm_NCLButtonDblClk

Beschreibung Diese Botschaft wird an ein Fenster übergeben, wenn der Anwender einen Doppelklick mit der linken Maustaste innerhalb eines Nicht-Client-Bereichs des Fensters ausführt.

Parameter *wParam:* Enthält den Code, der von *wm_NCHitTest* zurückgegeben wird. Weitere Informationen finden Sie in der Beschreibung der Botschaft *wm_NCHitTest*.

lParamLo: Die x-Koordinate des Mauszeigers.

lParamHi: Die y-Koordinate des Mauszeigers.

Rückgabewert Wird nicht verwendet.

Hinweis Die Koordinaten sind relativ zur oberen linken Ecke des Fensters.

Windows erzeugt eine Doppelklickbotschaft, wenn eine Maustaste gedrückt, losgelassen und innerhalb der vom System vorgegebenen Doppelklickzeit ein zweites Mal betätigt wird. Ein Doppelklick erzeugt vier Botschaften: Drücken der Taste, Loslassen der Taste, Doppelklickbotschaft, erneutes Loslassen der Taste.

Die voreingestellte Aktion von *DefWindowProc* beinhaltet das Senden von geeigneten *wm_SysCommand*-Botschaften (in Abhängigkeit vom betroffenen Nicht-Client-Bereich).

Vergleichen Sie auch die Beschreibungen folgender Botschaften: *wm_LButtonDblClk, wm_LButtonDown, wm_LButtonUp, wm_MButtonDblClk, wm_MButtonDown, wm_MButtonUp, wm_MouseMove, wm_NCLHitTest, wm_NCLButtonDown, wm_NCLButtonUp, wm_NCMButtonDblClk, wm_NCMButtonDown, wm_NCMButtonUp, wm_NCMouseMove, wm_NCRButtonDblClk, wm_NCRButtonDown, wm_NCRButtonUp, wm_RButtonDblClk, wm_RButtonDown* und *wm_RButtonUp*.

wm_NCLButtonDown

Beschreibung Diese Botschaft wird an ein Fenster übergeben, wenn der Anwender die linke Maustaste drückt, während sich der Mauszeiger im Nicht-Client-Bereich des Fensters befindet.

Parameter *wParam:* Enthält den Code, der von *wm_NCHitTest* zurückgegeben wird. Eine Beschreibung der Codes finden Sie bei der Botschaft *wm_NCHitTest*.

lParamLo: Die x-Koordinate des Mauszeigers.

lParamHi: Die y-Koordinate des Mauszeigers.

Rückgabewert Wird nicht verwendet.

Hinweis Die Koordinaten sind relativ zur oberen linken Ecke des Fensters.

Die voreingestellte Aktion von *DefWindowProc* beinhaltet das Senden von geeigneten *wm_SysCommand*-Botschaften (in Abhängigkeit vom betroffenen Nicht-Client-Bereich).

Vergleichen Sie auch die Beschreibungen folgender Botschaften: *wm_LButtonDblClk*, *wm_LButtonDown*, *wm_LButtonUp*, *wm_MButtonDblClk*, *wm_MButtonDown*, *wm_MButtonUp*, *wm_MouseMove*, *wm_NCLHitTest*, *wm_NCLButtonDblClk*, *wm_NCLButtonUp*, *wm_NCMButtonDblClk*, *wm_NCMButtonDown*, *wm_NCMButtonUp*, *wm_NCMouseMove*, *wm_NCRButtonDblClk*, *wm_NCRButtonDown*, *wm_NCRButtonUp*, *wm_RButtonDblClk*, *wm_RButtonDown* und *wm_RButtonUp*.

wm_NCLButtonUp

Beschreibung Diese Botschaft wird an ein Fenster übergeben, wenn der Anwender die linke Maustaste losläßt, während sich der Mauszeiger im Nicht-Client-Bereich des Fensters befindet.

Parameter *wParam:* Enthält den Code, der von *wm_NCHitTest* zurückgegeben wird. Eine Beschreibung des Codes finden Sie bei der Botschaft *wm_NCHitTest*.

lParamLo: Die x-Koordinate des Mauszeigers.

lParamHi: Die y-Koordinate des Mauszeigers.

Rückgabewert Wird nicht verwendet.

Hinweis Die Koordinaten sind relativ zur oberen linken Ecke des Fensters. Die voreingestellte Aktion von *DefWindowProc* beinhaltet das Senden von geeigneten *wm_SysCommand*-Botschaften (in Abhängigkeit vom betroffenen Nicht-Client-Bereich).

Vergleichen Sie auch die Beschreibungen folgender Botschaften: *wm_LButtonDblClk*, *wm_LButtonDown*, *wm_LButtonUp*, *wm_MButtonDblClk*, *wm_MButtonDown*, *wm_MButtonUp*, *wm_MouseMove*, *wm_NCLHitTest*, *wm_NCLButtonDblClk*, *wm_NCLButtonDown*, *wm_NCMButtonDblClk*, *wm_NCMButtonDown*, *wm_NCMButtonUp*, *wm_NCMouseMove*, *wm_NCRButtonDblClk*, *wm_NCRButtonDown*, *wm_NCRButtonUp*, *wm_RButtonDblClk*, *wm_RButtonDown* und *wm_RButtonUp*.

wm_NCMButtonDblClk

Beschreibung Diese Botschaft wird an ein Fenster übergeben, wenn der Anwender einen Doppelklick mit der mittleren Maustaste innerhalb des Nicht-Client-Bereichs des Fensters ausführt.

Parameter *wParam:* Enthält den Code, der von *wm_NCHitTest* zurück-

gegeben wird. Eine Beschreibung des Codes finden Sie bei der Botschaft *wm_NCHitTest*.

lParamLo: Die x-Koordinate des Mauszeigers.

lParamHi: Die y-Koordinate des Mauszeigers.

Rückgabewert Wird nicht verwendet.

Hinweis Die Koordinaten sind relativ zur oberen linken Ecke des Fensters.

Windows erzeugt eine Doppelklickbotschaft, wenn eine Maustaste gedrückt, losgelassen und dann innerhalb der vom System vorgegebenen Doppelklickzeit ein zweites Mal betätigt wird. Ein Doppelklick erzeugt vier Botschaften: Drücken der Taste, Loslassen der Taste, Doppelklickbotschaft, erneutes Loslassen der Taste.

Die voreingestellte Aktion von *DefWindowProc* beinhaltet das Senden von geeigneten *wm_SysCommand*-Botschaften (in Abhängigkeit vom betroffenen Nicht-Client-Bereich).

Vergleichen Sie auch die Beschreibungen folgender Botschaften: *wm_LButtonDblClk, wm_LButtonDown, wm_LButtonUp, wm_MButtonDblClk, wm_MButtonDown, wm_MButtonUp, wm_MouseMove, wm_NCLHitTest, wm_NCLButtonDblClk, wm_NCLButtonDown, wm_NCLButtonUp, wm_NCMButtonDown, wm_NCMButtonUp, wm_NCMouseMove, wm_NCRButtonDblClk, wm_NCRButtonDown, wm_NCRButtonUp, wm_RButtonDblClk, wm_RButtonDown* und *wm_RButtonUp*.

wm_NCMButtonDown

Beschreibung Diese Botschaft wird an das Fenster übergeben, wenn der Anwender die mittlere Maustaste drückt, während sich der Mauszeiger innerhalb eines Nicht-Client-Bereichs des Fensters befindet.

Parameter *wParam:* Einer der Codes, die von *wm_NCHitTest* zurückgegeben werden. Die Codes sind in der Botschaft *wm_NCHitTest* beschrieben.

lParamLo: Die x-Koordinate des Mauszeigers.

lParamHi: Die y-Koordinate des Mauszeigers.

Rückgabewert Wird nicht verwendet.

Hinweis Die Koordinaten sind relativ zur oberen linken Ecke des Fensters.

Die voreingestellte Aktion von *DefWindowProc* beinhaltet das Senden von geeigneten *wm_SysCommand*-Botschaften (in Abhängigkeit vom betroffenen Nicht-Client-Bereich).

Vergleichen Sie auch die Beschreibungen folgender Bot-

schaften: *wm_LButtonDblClk*, *wm_LButtonDown*, *wm_LButtonUp*, *wm_MButtonDblClk*, *wm_MButtonDown*, *wm_MButtonUp*, *wm_MouseMove*, *wm_NCLHitTest*, *wm_NCLButtonDblClk*, *wm_NCLButtonDown*, *wm_NCLButtonUp*, *wm_NCMButtonDblClk*, *wm_NCMButtonUp*, *wm_NCMouseMove*, *wm_NCRButtonDblClk*, *wm_NCRButtonDown*, *wm_NCRButtonUp*, *wm_RButtonDblClk*, *wm_RButtonDown* und *wm_RButtonUp*.

wm_NCMButtonUp

Beschreibung Diese Botschaft wird an das Fenster übergeben, wenn der Anwender die mittlere Maustaste losläßt, während sich der Mauszeiger innerhalb eines Nicht-Client-Bereichs des Fensters befindet.

Parameter *wParam:* Einer der Codes, die von *wm_NCHitTest* zurückgegeben werden. Die Codes sind in der Botschaft *wm_NCHitTest* beschrieben.

lParamLo: Die x-Koordinate des Mauszeigers.

lParamHi: Die y-Koordinate des Mauszeigers.

Rückgabewert Wird nicht verwendet.

Hinweis Die Koordinaten sind relativ zur oberen linken Ecke des Fensters. Die voreingestellte Aktion von *DefWindowProc* beinhaltet das Senden von geeigneten *wm_SysCommand*-Botschaften (in Abhängigkeit vom betroffenen Nicht-Client-Bereich).

Vergleichen Sie auch die Beschreibungen folgender Botschaften: *wm_LButtonDblClk*, *wm_LButtonDown*, *wm_LButtonUp*, *wm_MButtonDblClk*, *wm_MButtonDown*, *wm_MButtonUp*, *wm_MouseMove*, *wm_NCLHitTest*, *wm_NCLButtonDblClk*, *wm_NCLButtonDown*, *wm_NCLButtonUp*, *wm_NCMButtonDblClk*, *wm_NCMButtonDown*, *wm_NCMouseMove*, *wm_NCRButtonDblClk*, *wm_NCRButtonDown*, *wm_NCRButtonUp*, *wm_RButtonDblClk*, *wm_RButtonDown* und *wm_RButtonUp*.

wm_NCMouseMove

Beschreibung Diese Botschaft wird an ein Fenster übergeben, wenn der Mauszeiger innerhalb des Nicht-Client-Bereichs des Fensters bewegt wird.

Parameter *wParam:* Einer der Codes, die von *wm_NCHitTest* zurückgegeben werden. Die Codes sind in der Botschaft *wm_NCHitTest* beschrieben.

lParamLo: Die x-Koordinate des Mauszeigers.

lParamHi: Die y-Koordinate des Mauszeigers.

Rückgabewert Wird nicht verwendet.

Hinweis Die Koordinaten sind relativ zur oberen linken Ecke des Fensters.

Die voreingestellte Aktion von *DefWindowProc* beinhaltet das Senden von geeigneten *wm_SysCommand*-Botschaften (in Abhängigkeit vom betroffenen Nicht-Client-Bereich).

Vergleichen Sie auch die Beschreibungen folgender Botschaften: *wm_LButtonDblClk, wm_LButtonDown, wm_LButtonUp, wm_MButtonDblClk, wm_MButtonDown, wm_MButtonUp, wm_MouseMove, wm_NCLHitTest, wm_NCLButtonDblClk, wm_NCLButtonDown, wm_NCLButtonUp, wm_NCMButtonDblClk, wm_NCMButtonDown, wm_NCMButtonUp, wm_NCRButtonDblClk, wm_NCRButtonDown, wm_NCRButtonUp, wm_RButtonDblClk, wm_RButtonDown* und *wm_RButtonUp*.

wm_NCPaint

Beschreibung Diese Botschaft wird übergeben, wenn der Fensterrahmen gezeichnet werden muß.

Parameter *wParam:* Wird nicht verwendet.

lParam: Wird nicht verwendet.

Rückgabewert Wird nicht verwendet.

Hinweis Die Funktion *DefWindowProc* zeichnet den Fensterrahmen. Anwendungen können diese Botschaft abfangen und individuelle Fensterrahmen malen. Der Clipping-Bereich eines Fensters ist immer rechteckig, auch wenn die Form des Rahmens hiervon abweicht.

wm_NCRButtonDblClk

Beschreibung Diese Botschaft wird an das Fenster übergeben, wenn der Anwender einen Doppelklick mit der rechten Maustaste ausführt, während sich der Mauszeiger innerhalb eines Nicht-Client-Bereichs des Fensters befindet.

Parameter *wParam:* Einer der Codes, die von *wm_NCHitTest* zurückgegeben werden. Die Codes sind in der Botschaft *wm_NCHitTest* beschrieben.

lParamLo: Die x-Koordinate des Mauszeigers.

lParamHi: Die y-Koordinate des Mauszeigers.

Rückgabewert Wird nicht verwendet.

Hinweis Die Koordinaten sind relativ zur oberen linken Ecke des Fensters.

Windows erzeugt eine Doppelklickbotschaft, wenn eine Maustaste gedrückt, losgelassen und innerhalb der vom System vorgegebenen Doppelklickzeit ein zweites Mal betätigt wird. Ein Doppelklick erzeugt vier Botschaften: Drücken der Taste, Loslassen der Taste, Doppelklickbotschaft, erneutes Loslassen der Taste.

Die voreingestellte Aktion von *DefWindowProc* beinhaltet das Senden von geeigneten *wm_SysCommand*-Botschaften (in Abhängigkeit vom betroffenen Nicht-Client-Bereich).

Vergleichen Sie auch die Beschreibungen folgender Botschaften: *wm_LButtonDblClk*, *wm_LButtonDown*, *wm_LButtonUp*, *wm_MButtonDblClk*, *wm_MButtonDown*, *wm_MButtonUp*, *wm_MouseMove*, *wm_NCLHitTest*, *wm_NCLButtonDblClk*, *wm_NCLButtonDown*, *wm_NCLButtonUp*, *wm_NCMButtonDblClk*, *wm_NCMButtonDown*, *wm_NCMButtonUp*, *wm_NCMouseMove*, *wm_NCRButtonDown*, *wm_NCRButtonUp*, *wm_RButtonDblClk*, *wm_RButtonDown* und *wm_RButtonUp*.

wm_NCRButtonDown

Beschreibung

Diese Botschaft wird an das Fenster übergeben, wenn der Anwender die rechte Maustaste drückt, während sich der Mauszeiger innerhalb eines Nicht-Client-Bereichs des Fensters befindet.

Parameter

wParam: Einer der Codes, die von *wm_NCHitTest* zurückgegeben werden. Die Codes sind in der Botschaft *wm_NCHitTest* beschrieben.

lParamLo: Die x-Koordinate des Mauszeigers.

lParamHi: Die y-Koordinate des Mauszeigers.

Rückgabewert

Wird nicht verwendet.

Hinweis

Die Koordinaten sind relativ zur oberen linken Ecke des Fensters.

Die voreingestellte Aktion von *DefWindowProc* beinhaltet das Senden von geeigneten *wm_SysCommand*-Botschaften (in Abhängigkeit vom betroffenen Nicht-Client-Bereich).

Vergleichen Sie auch die Beschreibungen folgender Botschaften: *wm_LButtonDblClk*, *wm_LButtonDown*, *wm_LButtonUp*, *wm_MButtonDblClk*, *wm_MButtonDown*, *wm_MButtonUp*, *wm_MouseMove*, *wm_NCLHitTest*, *wm_NCLButtonDblClk*, *wm_NCLButtonDown*, *wm_NCLButtonUp*, *wm_NCMButtonDblClk*, *wm_NCMButtonDown*, *wm_NCMButtonUp*, *wm_NCMouseMove*, *wm_NCRButtonDblClk*, *wm_NCRButtonUp*, *wm_RButtonDblClk*, *wm_RButtonDown* und *wm_RButtonUp*.

wm_NCRButtonUp

Beschreibung Diese Botschaft wird an das Fenster übergeben, wenn der Anwender die rechte Maustaste losläßt, während sich der Mauszeiger innerhalb eines Nicht-Client-Bereichs des Fensters befindet.

Parameter *wParam:* Einer der Codes, die von *wm_NCHitTest* zurückgegeben werden. Die Codes sind in der Botschaft *wm_NCHitTest* beschrieben.

lParamLo: Die x-Koordinate des Mauszeigers.

lParamHi: Die y-Koordinate des Mauszeigers.

Rückgabewert Wird nicht verwendet.

Hinweis Die Koordinaten sind relativ zur oberen linken Ecke des Fensters. Die voreingestellte Aktion von *DefWindowProc* beinhaltet das Senden von geeigneten *wm_SysCommand*-Botschaften (in Abhängigkeit vom betroffenen Nicht-Client-Bereich).

Vergleichen Sie auch die Beschreibungen folgender Botschaften: *wm_LButtonDblClk, wm_LButtonDown, wm_LButtonUp, wm_MButtonDblClk, wm_MButtonDown, wm_MButtonUp, wm_MouseMove, wm_NCLHitTest, wm_NCLButtonDblClk, wm_NCLButtonDown, wm_NCLButtonUp, wm_NCMButtonDblClk, wm_NCMButtonDown, wm_NCMButtonUp, wm_NCMouseMove, wm_NCRButtonDblClk, wm_NCRButtonDown, wm_RButtonDblClk, wm_RButtonDown* und *wm_RButtonUp*.

wm_NextDlgCtl

Beschreibung Diese Botschaft wird an die Fensterfunktion eines Dialogfensters übergeben, um den Fokus einem anderen Steuerelement zu gehen. Das Resultat dieser Botschaft unterscheidet sich vom Resultat der Funktion *SetFocus* dadurch, daß die Botschaft *wm_NextDlgCtl* die Rahmenlinie um den Standardaktionsschalter modifiziert.

Parameter *wParam:* Ist *lParam* ungleich 0, bezeichnet *wParam* das Steuerelement, das den Fokus erhält. Ist *lParam* 0, stellt *wParam* ein Flag dar, das anzeigt, ob das nächste oder das vorhergehende Steuerelement den Fokus erhält. Ist *wParam* 0, erhält das nächste Steuerelement den Fokus, andernfalls das vorhergehende.

lParam: Enthält ein Flag, das anzeigt, wie Windows *wParam* interpretiert. Ist *lParam* ungleich 0, so stellt *wParam* ein Handle zum Steuerelement bereit, das den Fokus erhält. Andernfalls ist *wParam* ein Flag, das anzeigt, ob das nächste oder das vorhergehende Steuerelement den Fokus erhält.

Rückgabewert Wird nicht verendet.

Hinweis Die Funktion *SendMessage* sollte diese Botschaft keinesfalls übergeben, wenn die Anwendung gleichzeitig andere Botschaften zum Setzen des Fokus bearbeitet. Stattdessen kann die Funktion *PostMessage* verwendet werden.

wm_Paint

Beschreibung Diese Botschaft wird übergeben, wenn Windows oder eine Anwendung die Aktualisierung eines Teils eines Anwendungsfensters anfordert. Die Botschaft wird entweder beim Aufruf der Prozedur *UpdateWindow* oder durch die Funktion *DispatchMessage* übergeben, wenn die Anwendung die *wm_Paint*-Botschaft durch Verwendung der Funktionen *GetMessage* oder *PeekMessage* erhält.

Parameter *wParam:* Wird nicht verwendet.

lParam: Wird nicht verwendet.

Rückgabewert Wird nicht verwendet.

Hinweis Lesen Sie hierzu auch die Beschreibung der Routinen *BeginPaint* und *EndPaint*.

wm_PaintClipboard

Beschreibung Diese Botschaft wird übergeben, wenn die Zwischenablage ein Daten-Handle im Format *cf_OwnerDisplay* enthält (inbesondere sollte der Besitzer der Zwischenablage den Inhalt darstellen) und der Client-Bereich der Zwischenablage neu gezeichnet werden muß. Die Botschaft *wm_PaintClipboard* wird an den Besitzer der Zwischenablage übergeben, um die Aktualisierung des Client-Bereichs (in Teilen oder komplett) der Zwischenablage anzufordern.

Parameter *wParam:* Enthält ein Handle zum Fenster der Zwischenablage.

lParamLo: Ein Speicher-Handle zur *TPaintStruct*-Struktur, die den Teil der Zwischenablage definiert, der aktualisiert werden soll.

lParamHi: Wird nicht verwendet.

Rückgabewert Wird nicht verwendet.

Hinweis Um zu ermitteln, ob der gesamte Client-Bereich oder nur ein Teil aktualisiert werden muß, muß der Besitzer der Zwischenablage die Dimensionen des zu zeichnenden Bereichs im Feld *rcpaint* der *TPaintStruct*-Datenstruktur mit denen durch die Botschaft *wm_SizeClipboard* vorgegebenen vergleichen.

Anwendungen müssen die Funktion *GlobalLock* verwenden, um den Speicherbereich, der die *TPaintStruct*-Datenstruktur enthält, zu sperren. Dieser Speicherbereich sollte mit Hilfe der Funktion *GlobalUnlock* vor Rück- oder Abgabe der Steuerung wieder freigegeben werden.

Daten und Format der Zwischenablage werden mit der Funktion *SetClipboardData* gesetzt.

wm_PaintIcon

Beschreibung Diese Botschaft wird an ein zum Symbol verkleinertes Fenster übergeben, wenn das Symbol gezeichnet werden soll. Die Botschaft kann nur von Fenstern empfangen werden, für die ein Klassensymbol vereinbart ist. Andernfalls wird die Botschaft *wm_Paint* übergeben. Die Übergabe dieser Botschaft an die Funktion *DefWindowProc* ermöglicht es Windows, das Symbol mit Hilfe des Klassensymbols zu zeichnen.

Parameter *wParam:* Wird nicht verwendet.

lParam: Wird nicht verwendet.

Rückgabewert Wird nicht verwendet.

wm_PaletteChanged

Beschreibung Diese Botschaft teilt allen Fenstern mit, daß das Fenster mit dem Fokus bei der Realisierung seiner virtuellen Farbskala die Systemfarbskala verändert hat. Dies erlaubt es Fenstern ohne Fokus ihre virtuelle Farbskala zu realisieren und ihre Client-Bereiche zu aktualisieren.

Parameter *wParam:* Enthält das Handle des Fensters, das die Systemfarbskala verändert hat.

lParam: Wird nicht verwendet.

Rückgabewert Wird nicht verwendet.

Hinweis Um die Programmierung einer Endlosschleife zu vermeiden, sollte ein Fenster vor Realisierung der eigenen Farbskala sicherstellen, daß der Parameter *wParam* nicht sein eigenes Handle enthält.

wm_ParentNotify

Beschreibung Diese Botschaft wird an das übergeordnete Fenster übergeben, wenn das untergeordnete Fenster erstellt oder entfernt wird, oder wenn der Anwender eine Maustaste gedrückt hat, während der Mauszeiger über einem untergeordneten Fenster liegt.

Wird das untergeordnete Fenster erstellt, übergibt Windows die Botschaft *wm_ParentNotify* unmittelbar vor Rückkehr der erzeugenden *CreateWindow*- bzw. *CreateWindowEx*-Funktion. Soll das untergeordnete Fenster entfernt werden, wird die Botschaft von Windows noch vor Entfernung des Fensters übergeben.

Parameter *wParam:* Bestimmt das dem übergeordneten Fenster mitzuteilende Ereignis. Möglich sind folgende Werte:

Wert	Bedeutung
wm_Create	Das untergeordnete Fenster wird gerade erzeugt.
wm_Destroy	Das untergeordnete Fenster wird gerade entfernt.
wm_LButtonDown	Der Anwender hat ein untergeordnetes Fenster angeklickt.
wm_MButtonDown	Der Anwender hat ein untergeordnetes Fenster angeklickt.
wm_RButtonDown	Der Anwender hat ein untergeordnetes Fenster angeklickt.

Parameter *lParamLo:* Das Handle des untergeordneten Fensters, wenn *wParam wm_Create* oder *wm_Destroy* ist. Andernfalls enthält *lParamLo* die x-Koordinate des Mauszeigers.

lParamHi: Die ID des untergeordneten Fensters, wenn *wParam wm_Create* oder *wm_Destroy* ist. Andernfalls enthält *lParamHi* die y-Koordinate des Mauszeigers.

Rückgabewert Wird nicht verwendet.

Hinweis Diese Botschaft wird an alle einem bestimmten Fenster übergeordneten Fenster bis hin zum Hauptfenster übergeben. Die Botschaft wird an die übergeordneten Fenster aller untergeordneten Fenster übergeben, außer das untergeordnete Fenster ist vom erweiterten Fensterstil *ws_ex_NoParentNotify*. Die Funktion *CreateWindowEx* erzeugt Fenster mit erweiterten Fensterstilen. In der Standardeinstellung sind alle einem Dialogfenster untergeordneten Fenster vom Stil *ws_ex_NoParentNotify*.

wm_Paste

Beschreibung Diese Botschaft fügt die Daten aus der Zwischenablage in das Fenster an der aktuellen Mauszeigerposition ein. Die Daten werden nur dann eingefügt, wenn sie in der Zwischenablage im Format *cf_Text* vorliegen.

Parameter *wParam:* Wird nicht verwendet.

lParam: Wird nicht verwendet.

Rückgabewert Wird nicht verwendet.

wm_QueryDragIcon

Beschreibung Diese Botschaft wird an ein zum Symbol verkleinertes Fenster übergeben, das vom Anwender verschoben wird und das kein definiertes Klassensymbol besitzt. Verschiebt der Anwender das Symbol eines Fensters ohne Klassensymbol, ersetzt Windows dieses Symbol durch den Standardsymbolzeiger. Benötigt die Anwendung einen anderen Mauszeiger, um während des Verschiebens angezeigt zu werden, muß sie das Handle eines monochromen Mauszeigers zurückgeben. Der Mauszeiger muß zur Auflösungsmöglichkeit des Gerätetreibers passen.

Die Anwendung kann die Funktion *LoadCursor* aufrufen, um einen Mauszeiger aus den Ressourcen ihrer ausführbaren Datei zu laden und dessen Handle abzurufen.

Parameter *wParam:* Wird nicht verwendet.

lParam: Wird nicht verwendet.

Rückgabewert Der Rückgabewert enthält im niederwertigen Word das Handle des Mauszeigers, den Windows während des Verschiebens darstellen soll. Er ist 0, wenn Windows den Standardsymbolzeiger darstellen soll. Der Standardrückgabewert ist 0.

wm_QueryEndSession

Beschreibung Diese Botschaft wird übergeben, wenn der Anwender den Befehl zum Beenden der Arbeitssitzung anwählt. Gibt eine Anwendung 0 zurück, wird die Arbeitssitzung nicht beendet. In diesem Augenblick wird auch die weitere Übergabe von *wm_QueryEndSession*-Botschaften durch Windows beendet und allen Anwendungen, die einen Wert ungleich 0 zurückgegeben haben, wird die Botschaft *wm_EndSession* mit dem auf 0 gesetzten Parameter *wParam* übergeben.

Parameter *wParam:* Wird nicht verwendet.

lParam: Wird nicht verwendet.

Rückgabewert Der Rückgabewert ist ungleich 0, wenn die Anwendung problemlos geschlossen werden kann. Andernfalls ist der Rückgabewert 0.

Hinweis Die Funktion *DefWindowProc* gibt standardmäßig einen Wert ungleich 0 zurück.

wm_QueryNewPalette

Beschreibung Diese Botschaft teilt einem Fenster mit, daß es den Fokus erhalten wird. Realisiert das Fenster beim Empfang des Fokus

eine eigene virtuelle Farbskala, sollte es einen Wert ungleich 0, andernfalls 0 zurückgeben.

Parameter *wParam:* Wird nicht verwendet.

lParam: Wird nicht verwendet.

Rückgabewert Der Rückgabewert ist ungleich 0, wenn das Fenster seine virtuelle Farbskala realisiert. Andernfalls wird 0 zurückgegeben.

wm_QueryOpen

Beschreibung Diese Botschaft wird an ein Symbol übergeben, wenn der Anwender dessen Öffnung zu einem Fenster anfordert.

Parameter *wParam:* Wird nicht verwendet.

lParam: Wird nicht verwendet.

Rückgabewert Der Rückgabewert ist 0, wenn die Anwendung die Öffnung des Symbols verweigert und ungleich 0, wenn das Symbol geöffnet werden kann.

Hinweis Die Funktion *DefWindowProc* gibt einen Wert ungleich 0 zurück.

wm_Quit

Beschreibung Diese Botschaft zeigt die Aufforderung zum Beenden einer Anwendung an. Sie wird erzeugt, wenn die Anwendung die Funktion *PostQuitMessage* aufruft. Die Funktion *GetMessage* gibt daraufhin 0 zurück.

Parameter *wParam:* Enthält den im Funktionsaufruf von *PostQuitMessage* vorgegebenen Exit-Code.

lParam: Wird nicht verwendet.

Rückgabewert Aufrufe der Funktion *GetMessage* geben 0 zurück, nachdem diese Botschaft übergeben wurde. Der Exit-Code in *wParam* muß gesichert und als Exit-Code der Anwendung verwendet werden.

wm_RButtonDblClk

Beschreibung Diese Botschaft tritt auf, wenn der Anwender einen Doppelklick mit der rechten Maustaste ausführt.

Parameter *wParam:* Enthält einen Wert, der den Status verschiedener virtueller Tasten anzeigt. Jede Kombination der folgenden Werte ist möglich:

Wert	Bedeutung
mk_Control	Gesetzt, wenn die Ctrl -Taste gedrückt wird.
mk_LButton	Gesetzt, wenn die linke Maustaste gedrückt wird.
mk_MButton	Gesetzt, wenn die mittlere Maustaste gedrückt wird.
mk_RButton	Gesetzt, wenn die rechte Maustaste gedrückt wird.
mk_Shift	Gesetzt, wenn die Shift -Taste gedrückt wird.

lParamLo: Die x-Koordinate des Mauszeigers.

lParamHi: Die y-Koordinate des Mauszeigers.

Rückgabewert Wird nicht verwendet.

Hinweis Die Koordinaten sind relativ zur oberen linken Ecke des Fensters.

Windows erzeugt eine Doppelklickbotschaft, wenn eine Maustaste gedrückt, losgelassen und innerhalb der vom System vorgegebenen Doppelklickzeit ein zweites Mal betätigt wird. Ein Doppelklick erzeugt vier Botschaften: Drücken der Taste, Loslassen der Taste, Doppelklickbotschaft, erneutes Loslassen der Taste. Die voreingestellte Aktion von *DefWindowProc* beinhaltet das Senden von geeigneten *wm_SysCommand*-Botschaften (in Abhängigkeit vom betroffenen Nicht-Client-Bereich).

Vergleichen Sie auch die Beschreibungen folgender Botschaften: *wm_LButtonDblClk, wm_LButtonDown, wm_LButtonUp, wm_MButtonDblClk, wm_MButtonDown, wm_MButtonUp, wm_MouseMove, wm_NCLHitTest, wm_NCLButtonDblClk, wm_NCLButtonDown, wm_NCLButtonUp, wm_NCMButtonDblClk, wm_NCMButtonDown, wm_NCMButtonUp, wm_NCMouseMove, wm_NCRButtonDblClk, wm_NCRButtonDown, wm_NCRButtonUp, wm_RButtonDown* und *wm_RButtonUp*.

wm_RButtonDown

Beschreibung Diese Botschaft tritt auf, wenn der Anwender die rechte Maustaste drückt.

Parameter *wParam:* Enthält einen Wert, der den Status verschiedener virtueller Tasten anzeigt. Jede Kombination der folgenden Werte ist möglich:

Wert	Bedeutung
mk_Control	Gesetzt, wenn die Ctrl -Taste gedrückt wird.
mk_LButton	Gesetzt, wenn die linke Maustaste gedrückt wird.
mk_MButton	Gesetzt, wenn die mittlere Maustaste gedrückt wird.
MK-Shift	Gesetzt, wenn die Shift -Taste gedrückt wird.

lParamLo: Die x-Koordinate des Mauszeigers.

lParamHi: Die y-Koordinate des Mauszeigers.

Rückgabewert Wird nicht verwendet.

Hinweis Die Koordinaten sind relativ zur oberen linken Ecke des Fensters.

Die voreingestellte Aktion von *DefWindowProc* beinhaltet das Senden von geeigneten *wm_SysCommand*-Botschaften (in Abhängigkeit vom betroffenen Nicht-Client-Bereich).

Vergleichen Sie auch die Beschreibungen folgender Botschaften: *wm_LButtonDblClk*, *wm_LButtonDown*, *wm_LButtonUp*, *wm_MButtonDblClk*, *wm_MButtonDown*, *wm_MButtonUp*, *wm_MouseMove*, *wm_NCLHitTest*, *wm_NCLButtonDblClk*, *wm_NCLButtonDown*, *wm_NCLButtonUp*, *wm_NCMButtonDblClk*, *wm_NCMButtonDown*, *wm_NCMButtonUp*, *wm_NCMouseMove*, *wm_NCRButtonDblClk*, *wm_NCRButtonDown*, *wm_NCRButtonUp*, *wm_RButtonDblClk* und *wm_RButtonUp*.

wm_RButtonUp

Beschreibung Diese Botschaft tritt auf, wenn der Anwender die rechte Maustaste losläßt.

Parameter *wParam:* Enthält einen Wert, der den Status verschiedener virtueller Tasten anzeigt. Jede Kombination der folgenden Werte ist möglich:

Wert	Bedeutung
mk_Control	Gesetzt, wenn die Ctrl -Taste gedrückt wird.
mk_LButton	Gesetzt, wenn die linke Maustaste gedrückt wird.
mk_MButton	Gesetzt, wenn die mittlere Maustaste gedrückt wird.
mk_Shift	Gesetzt, wenn die Shift -Taste gedrückt wird.

lParamLo: Die x-Koordinate des Mauszeigers.

lParamHi: Die y-Koordinate des Mauszeigers.

Rückgabewert Wird nicht verwendet.

Hinweis Die Koordinaten sind relativ zur oberen linken Ecke des Fensters.

Die voreingestellte Aktion von *DefWindowProc* beinhaltet das Senden von geeigneten *wm_SysCommand*-Botschaften (in Abhängigkeit vom betroffenen Nicht-Client-Bereich).

Vergleichen Sie auch die Beschreibungen folgender Botschaften: *wm_LButtonDblClk*, *wm_LButtonDown*, *wm_LButtonUp*, *wm_MButtonDblClk*, *wm_MButtonDown*, *wm_MButtonUp*, *wm_MouseMove*, *wm_NCLHitTest*, *wm_NCLButtonDblClk*, *wm_NCLButtonDown*, *wm_NCLButtonUp*, *wm_NCMButtonDblClk*,

wm_NCMButtonDown, wm_NCMButtonUp, wm_NCMouseMove, wm_NCRButtonDblClk, wm_NCRButtonDown, wm_NCRButtonUp, wm_RButtonDblClk und *wm_RButtonDown.*

wm_RenderAllFormats

Beschreibung Diese Botschaft wird an eine zu entfernende Anwendung übergeben, wenn diese die Zwischenablage besitzt.

Parameter *wParam:* Wird nicht verwendet.

lParam: Wird nicht verwendet.

Rückgabewert Wird nicht verwendet.

Hinweis Die Anwendung sollte die Daten der Zwischenablage mit allen Formaten zurückgeben, die sie erzeugen kann. Dabei muß ein Handle für jedes Format an die Funktion *SetClipboardData* übergeben werden. Dies stellt sicher, daß die Daten in der Zwischenablage auch nach Beendigung der Anwendung bearbeitet werden können.

wm_RenderFormat

Beschreibung Durch diese Botschaft wird der Besitzer der Zwischenablage aufgefordert, die letzten in die Zwischenablage kopierten Daten im angegebenen Format zu formatieren. Anschließend wird das Handle der formatierten Daten an die Zwischenablage übergeben.

Parameter *wParam:* Bestimmt das Datenformat. Möglich sind die bei der Funktion *SetClipboardData* beschriebenen Formate.

lParam: Wird nicht verwendet.

Rückgabewert Wird nicht verwendet.

wm_SetCursor

Beschreibung Diese Botschaft tritt auf, wenn kein Fenster den Mausfang besitzt, der Mauszeiger aber in einem Fenster bewegt wird.

Parameter *wParam:* Enthält das Handle des Fensters, in dem sich der Mauszeiger befindet.

lParamLo: Eine der *ht*-Konstanten, die in Kapitel 1 »Windows-Stile und -Konstanten« im Abschnitt »htXxxx« beschrieben sind. Diese Werte werden auch von *wm_NCHiTest* zurückgegeben.

lParamHi: Die Nummer der Mausbotschaft.

Rückgabewert Wird 0 zurückgegeben, führt *DefWindowProc* seine Standardaktionen aus. Andernfalls finden keine weiteren Aktionen statt.

Hinweis Vor Bearbeitung der Funktion *DefWindowProc* übergibt sie die Botschaft *wm_SetCursor* an das übergeordnete Fenster. Gibt dieses Fenster einen Wert ungleich 0 zurück, wird die weitere Bearbeitung gestoppt. Die Übergabe der Botschaft an ein übergeordnetes Fenster ermöglicht diesem die Kontrolle des Mauszeigers in einem untergeordneten Fenster. Die Funktion *DefWindowProc* verwendet diese Botschaft außerdem, um dem Mauszeiger die Pfeilform zuzuweisen, wenn er sich in keinem Client-Bereich befindet. Andernfalls wird die registrierte klassenspezifische Form zugewiesen. Hat der Parameter *lParamLo* den Wert *htError* und enthält *lParamHi* eine Maustastendruck-Botschaft, wird die Funktion *MessageBeep* aufgerufen. *lParamHi* ist 0, wenn das Fenster in den Menümodus wechselt.

wm_SetFocus

Beschreibung Diese Botschaft wird übergeben, wenn ein Fenster den Fokus erhält.

Parameter *wParam:* Enthält das Handle des Fensters, das den Fokus abgibt (kann 0 sein).

lParam: Wird nicht verwendet.

Rückgabewert Wird nicht verwendet.

Hinweis Um ein Caret darzustellen, sollte die Anwendung an dieser Stelle die entsprechenden Funktionen aufrufen.

wm_SetFont

Beschreibung Diese Botschaft bestimmt die zur Textdarstellung verwendete Schrift eines Dialogfensters. Die Schrift sollte am besten gesetzt werden, wenn der Besitzer des Dialogfensters eine *wm_InitDialog*-Botschaft empfängt. Die Anwendung sollte nicht mehr benötigte Schriften mit der Funktion *DeleteObject* löschen, wenn zum Beispiel ein Steuerelement entfernt wird.

Diese Botschaft bewirkt keine Veränderung der Größe des Steuerelements. Um zu vermeiden, daß Windows Text einfügt, der nicht in die Umgrenzung eines Steuerelements paßt, sollte die Anwendung die Größe des Steuerfensters anpassen, bevor die Schrift geändert wird.

Parameter *wParam:* Enthält das Handle der Schrift. Hat dieser Parameter den Wert 0, stellt das Steuerelement den Text in der Standardsystemschrift dar.

lParam: Legt fest, ob das Steuerelement unmittelbar nach dem Setzen der Schrift neu gezeichnet werden soll. Ist *lParam* ungleich 0, wird das Steuerelement neu gezeichnet.

Rückgabewert Wird nicht verwendet.

Hinweis Bevor Windows ein Dialogfenster des Stils *ds_SetFont* erzeugt, wird die Botschaft *wm_SetFont* an das übergeordnete Fenster übergeben.

Eine Anwendung erzeugt ein Dialogfenster des Stils *ds_SetFont* durch Aufruf einer der folgenden Funktionen:

CreateDialogIndirect, CreateDialogIndirectParam, DialogBoxIndirect oder *DialogBoxIndirectParam*

wm_SetRedraw

Beschreibung Diese Botschaft wird von der Anwendung an ein Fenster übergeben, um die Darstellung von Veränderungen innerhalb des Fensters zuzulassen oder zu unterbinden.

Parameter *wParam:* Bestimmt den Status des Aktualisierungs-Flags. Ist der Parameter *wParam* ungleich 0, ist das Flag gesetzt.

lParam: Wird nicht verwendet.

Rückgabewert Wird nicht verwendet.

Hinweis Diese Botschaft setzt oder löscht das Aktualisierungs-Flag. Dieses bezieht sich allerdings nicht auf Listen. Ist das Aktualisierungs-Flag gesetzt, kann ein Steuerelement unmittelbar nach jeder Veränderung aktualisiert werden, sonst nicht. Anwendungen, die einer Liste mehrere Namen hinzufügen wollen, sollten das Flag vor dem Hinzufügen des letzten Namens setzen.

In diesem Fall wird erst unmittelbar nach dem Hinzufügen des letzten Namens die Neudarstellung der Liste veranlaßt.

wm_SetText

Beschreibung Diese Botschaft wird verwendet, um den Text eines Fensters zu setzen. Der zu setzende Text entspricht bei Editierfeldern und Editierfeldern in Kombinationsfenstern dem Inhalt der Editierfelder.

Bei Schaltern entspricht der Text dem Namen des Schalters. In allen anderen Fenstern ist der Text die Überschrift des Fensters.

Parameter *wParam:* Wird nicht verwendet.

lParam: Zeigt auf den nullterminierten String mit dem Fenstertext.

Rückgabewert Der Rückgabewert ist bei Listenfenstern *lb_ErrSpace* bzw. *cb_ErrSpace* bei Kombinationsfenstern, wenn der verfügbare Platz nicht ausreicht, um den Text in das Editierfeld zu setzen. Wird die Botschaft an ein Kombinationsfenster ohne Editierfeld übergeben, ist der Rückgabewert *cb_Err*.

Hinweis Durch diese Botschaft wird die aktuelle Auswahl in der Liste oder im Kombinationsfenster nicht verändert.

Mit Hilfe der Botschaft *cb_SelectString* kann eine Anwendung den Eintrag auswählen, der dem Text im Editierfeld entspricht.

wm_ShowWindow

Beschreibung Diese Botschaft wird übergeben, wenn ein Fenster gezeigt oder verborgen wird: nach Aufruf der Funktion *ShowWindow*, nach Aktualisierung oder Vergrößerung eines überlappenden Fensters oder nach dem Schließen (Symbol) bzw. Öffnen (Bildschirmdarstellung) eines überlappenden bzw. eines Pop-up-Fensters. Wird ein überlappendes Fenster geschlossen, werden alle zugehörigen Pop-up-Fenster verborgen.

Parameter *wParam:* Bestimmt, ob ein Fenster gezeigt wird. Ist der Wert 0, wird das Fenster verborgen. Ist er ungleich 0, wird es gezeigt.

lParam: Bestimmt den Status des zu zeigenden Fensters. Der Parameter *lParam* hat den Wert 0, wenn die Botschaft aufgrund eines Aufrufs der Funktion *ShowWindow* übergeben wird.

Andernfalls sind folgende Werte möglich:

Wert	Bedeutung
sw_ParentClosing	Es wird ein übergeordnetes Fenster geschlossen (Symbol) oder ein Pop-up-Fenster verborgen.
sw_ParentOpening	Es wird ein übergeordnetes Fenster geöffnet oder ein Pop-up-Fenster gezeigt.

Rückgabewert Wird nicht verwendet.

Hinweis Die Funktion *DefWindowProc* verbirgt oder zeigt das durch die Botschaft festgelegte Fenster.

wm_Size

Beschreibung Diese Botschaft wird nach Änderung der Größe eines Fensters übergeben.

Parameter *wParam:* Enthält einen Wert, der den Typ der gewünschten Größenänderung anzeigt. Möglich sind folgende Werte:

Wert	Bedeutung
SizeFullScreen	Fenster wurde zum Vollbild vergrößert.
SizeIconic	Fenster wurde zum Symbol verkleinert.
SizeNormal	Die Größe des Fensters wurde geändert, *SizeIconic* oder *SizeFullScreen* treffen jedoch nicht zu.
SizeZoomHide	Die Botschaft wird an alle Pop-up-Fenster übergeben, wenn ein anderes Fenster zum Vollbild vergrößert wurde.
SizeZoomShow	Die Botschaft wird an alle Pop-up-Fenster übergeben, wenn ein anderes Fenster in seiner voreingestellten Größe wiederhergestellt wurde.

Parameter *lParamLo:* Enthält die neue Breite des Client-Bereichs des Fensters.

lParamHi: Enthält die neue Höhe des Client-Bereichs des Fensters.

Rückgabewert Wird nicht verwendet.

Hinweis Wird als Ergebnis der Botschaft *wm_Size* eine der Routinen *SetScrollPos* oder *MoveWindow* für ein untergeordnetes Fenster aufgerufen, sollte der Parameter *Redraw* bzw. *Repaint* ungleich 0 sein, um das Fenster zur Aktualisierung der Darstellung zu veranlassen.

wm_SizeClipboard

Beschreibung Diese Botschaft wird übergeben, wenn die Zwischenablage ein Daten-Handle des Formats *cf_OwnerDisplay* (d.h. der Besitzer der Zwischenablage sollte den Inhalt der Zwischenablage darstellen) enthält und die Größe des Fensters der Zwischenablage geändert wurde.

Parameter *wParam:* Das Handle des Fensters der Zwischenablage.

lParamLo: Zeigt auf eine *TRect*-Datenstruktur, die den Bereich des Fensters der Zwischenablage bestimmt, der neu dargestellt werden soll.

lParamHi: Wird nicht verwendet.

Rückgabewert Wird nicht verwendet.

Hinweis Die *wm_SizeClipboard*-Botschaft wird mit einem Rechteck mit den Koordinaten (0,0,0,0) als neuer Größe übergeben, wenn die Zwischenablage entfernt oder verkleinert werden soll. Dies erlaubt es dem Besitzer der Zwischenablage, die Bildschirm-Ressourcen freizugeben. Anwendungen müssen die Funktion *GlobalLock* verwenden, um den Speicher zu sperren, der die

TPaintStruct-Datenstruktur enthält. Sie sollten den Speicher mit Hilfe der Funktion *GlobalUnlock* wieder freigeben, bevor sie die Steuerung ab- oder zurückgeben.

wm_SpoolerStatus

Beschreibung Diese Botschaft wird vom Druck-Manager übergeben, wenn ein Auftrag aus der Warteschlange des Druck-Managers entfernt oder in diese eingefügt wird.

Parameter *wParam:* Wird auf *pr_JobStatus* gesetzt.

lParamLo: Anzahl der Druckaufträge, die sich aktuell in der Warteschlange des Druck-Managers befinden.

lParamHi: Wird nicht verwendet.

Rückgabewert Wird nicht verwendet.

Hinweis Diese Botschaft dient nur zu Informationszwecken.

wm_SysChar

Beschreibung Diese Botschaft tritt auf, wenn *wm_SysKeyUp*- und *wm_SysKeyDown*-Botschaften übersetzt werden. Sie bestimmt den virtuellen Tastencode der Systemmenütaste.

Parameter *wParam:* Enthält den ASCII-Tastencode einer Systemmenütaste.

lParamLo: Wiederholungszähler. Der Wert ergibt sich aus der Zeit, die der Anwender die Taste gedrückt hält und entspricht der ermittelten Anzahl von Tastenwiederholungen.

lParamHi: Die Bits von *lParamHi* können folgende Werte annehmen:

Bit	Bedeutung
0 bis 7	Enthalten den Scan-Code der Taste (OEM-abhängiger Wert).
8	Ist 1, wenn die gedrückte Taste eine zusätzliche Taste (z.B. eine Funktionstaste) ist.
9 bis 12	Werden entweder überhaupt nicht oder nur intern von Windows verwendet.
13	Ist 1, wenn gleichzeitig mit der Taste die Alt -Taste gedrückt wurde.
14	Ist 1, wenn die Taste vor Übergabe dieser Botschaft bereits gedrückt war.
15	Ist 1, wenn die Taste losgelassen, 0, wenn sie gedrückt wurde.

Rückgabewert Wird nicht verwendet.

Hinweis Ist der Kontextcode 0, kann die Botschaft an die Funktion *TranslateAccelerator* weitergegeben werden. Diese behandelt die Botschaft wie eine normale Tastaturbotschaft, nicht wie eine Systemtastenbotschaft. Dies ermöglicht es, Acceleator-Tasten zusammen mit dem aktiven Fenster zu verwenden, auch wenn dieses nicht den Fokus hat.

Bei erweiterten IBM-Tastaturen mit 101 oder 102 Tasten gelten folgende Tasten als zusätzliche: die rechte Alt - und Ctrl -Taste im Hauptbereich der Tastatur, die Tasten Ins , Del , Home , End , PgUp , PgDown und die Cursor-Tasten im Block zwischen dem Nummernblock und der alphanumerischen Tastatur sowie die Divisions- und die Eingabetaste im Nummernblock. Auch andere Tastaturen unterstützen möglicherweise das Zusatztasten-Bit im Parameter *lParamHi*.

wm_SysColorChange

Beschreibung Diese Botschaft bestimmt den Wechsel einer oder mehrerer Systemfarben. Windows übergibt die Botschaft an alle Hauptfenster, wenn eine Änderung in der Vorgabe der Systemfarben stattfindet.

Parameter *wParam:* Wird nicht verwendet.

lParam: Wird nicht verwendet.

Rückgabewert Wird nicht verwendet.

Hinweis Windows übergibt eine *wm_Paint*-Botschaft an alle von der Systemfarbenänderung beeinflußten Fenster. Anwendungen, die Pinsel in den existierenden Systemfarben besitzen, sollten diese löschen und mit den neuen Systemfarben neu erstellen.

wm_SysCommand

Beschreibung Diese Botschaft wird übergeben, wenn der Anwender einen Befehl aus dem Systemmenü oder das Symbol für Vollbild- bzw. Symboldarstellung anwählt.

Parameter *wParam:* Bestimmt den Typ des angeforderten Systembefehls. Möglich ist einer der folgenden Werte:

Wert	Bedeutung
sc_Close	Fenster schließen.
sc_HScroll	Horizontalen Bildlauf durchführen.
sc_KeyMenu	Das dem Tastendruck entsprechende Menü darstellen.

Wert	Bedeutung
sc_Maximize, sc_Zoom	Fenster als Vollbild darstellen.
sc_Minimize sc_Icon	Fenster als Symbol darstellen.
sc_MouseMenu	Das dem Mausklick entsprechende Menü darstellen.
sc_Move	Das Fenster verschieben.
sc_NextWindow	In das nächste Fenster bewegen.
sc_PrevWindow	In das vorhergehende Fenster bewegen.
sc_Restore	Stellt nach einer Maximierung oder Minimierung die ursprüngliche Größe des Fensters wieder her
sc_Size	Größe des Fensters einstellen.
sc_VScroll	Vertikalen Bildlauf durchführen.

lParamLo: x-Koordinate des Mauszeigers oder 0, wenn die Maus nicht verwendet wurde.

lParamHi: y-Koordinate des Mauszeigers oder 0, wenn die Maus nicht verwendet wurde.

Rückgabewert Wird nicht verwendet.

Hinweis Die Anfrage an das Systemmenü wird von der Funktion *DefWindowProc*, wie oben festgelegt, ausgeführt.

In *wm_SysCommand*-Botschaften werden die vier niederwertigen Bits des Parameters *wParam* intern von Windows verwendet. Um den korrekten Wert von *wParam* zu ermitteln, müssen Anwendungen dem Wert des Parameters *wParam* den Wert $FFF0 hinzufügen, indem sie den bitweisen and-Operator verwenden.

Die Menüeinträge eines Systemmenüs können durch Verwendung der Funktionen *GetSystemMenu, AppendMenu, InsertMenu* und *ModifyMenu* modifiziert werden. Anwendungen, die das Systemmenü modifizieren, müssen die *wm_SysCommand*-Botschaften bearbeiten. Werden die Botschaften nicht bearbeitet, müssen sie an die Funktion *DefWindowProc* übergeben werden.

Durch die Anwendung hinzugefügte Befehlswerte können von dieser Funktion nicht verarbeitet werden, dies muß durch die Anwendung selbst geschehen. Anwendungen können jederzeit jeden Systembefehl ausführen, indem sie eine *wm_SysCommand*-Botschaft an die Funktion *DefWindowProc* übergeben. Accelerator-Tasten, die für die Auswahl eines Eintrags im Systemmenü definiert wurden, werden in *wm_SysCommand*-Botschaften, alle anderen in *wm_Command*-Botschaften übersetzt.

wm_SysDeadChar

Beschreibung Diese Botschaft resultiert aus der Übersetzung einer *wm_SysKeyUp*- und *wm_SysKeyDown*-Botschaft. Sie bestimmt den Wert einer toten Taste.

Parameter *wParam:* Enthält den Zeichenwert einer toten Taste.

lParamLo: Anzahl der Tastenbetätigungen.

lParamHi: Anzahl der Tastenwiederholungen, die aus der Dauer des Tastendrucks ermittelt wird.

Rückgabewert Wird nicht verwendet.

Hinweis Tote Tasten enthalten das Umlaut-Zeichen und Akzente. Diese Botschaft kann verwendet werden, um auf Tastenbetätigungen zu reagieren, die selber nicht unbedingt zur Darstellung eines Zeichens führen (siehe auch die Beschreibungen der Funktionen *wm_Char*, *wm_DeadChar* und *wm_SysChar*).

wm_SysKeyDown

Beschreibung Dieser Botschaft wird übergeben, wenn der Anwender die Alt-Taste gedrückt hält und gleichzeitig eine andere Taste betätigt. Sie tritt auch auf, wenn aktuell kein Fenster den Fokus hat. In diesem Fall wird die Botschaft *wm_SysKeyDown* an das aktive Fenster übergeben. Das empfangende Fenster kann zwischen den beiden Kontexten durch Überprüfung des Kontextcodes im Parameter *lParamHi* unterscheiden.

Parameter *wParam:* Enthält den virtuellen Tastencode der gedrückten Taste.

lParamLo: Wiederholungszähler. Der Wert ergibt sich aus der Zeit, die der Anwender die Taste gedrückt hält und entspricht der ermittelten Anzahl von Tastenwiederholungen.

lParamHi: Die Bits von *lParamHi* können folgende Werte annehmen:

Bit	Bedeutung
0 bis 7	Enthalten den Scan-Code der Taste (OEM-abhängiger Wert).
8	Ist 1, wenn die gedrückte Taste eine zusätzliche Taste (z.B. eine Funktionstaste) ist.
9 bis 12	Werden entweder überhaupt nicht oder nur intern von Windows verwendet.
13	Ist 1, wenn gleichzeitig mit der Taste die Alt-Taste gedrückt wurde.

Bit	Bedeutung
14	Ist 1, wenn die Taste vor Übergabe dieser Botschaft bereits gedrückt war.
15	Ist 1, wenn die Taste losgelassen, 0, wenn sie gedrückt wurde.

Rückgabewert Wird nicht verwendet.

Hinweis Ist der Kontextcode 0, kann die Botschaft an die Funktion *TranslateAccelerator* weitergegeben werden. Diese behandelt die Botschaft wie eine normale Tastaturbotschaft, nicht wie eine Systemtastenbotschaft. Dies ermöglicht es, Acceleator-Tasten zusammen mit dem aktiven Fenster zu verwenden, auch wenn dieses nicht den Fokus hat.

Wegen der automatischen Tastenwiederholung kann es zu mehreren *wm_SysKeyDown*-Botschaften kommen, bevor eine *wm_SysKeyUp*-Botschaft übergeben wird. Der vorhergehende Tastenstatus kann verwendet werden, um festzustellen, ob die Botschaft *wm_SysKeyDown* einen ersten oder einen wiederholten Tastendruck anzeigt.

Bei erweiterten IBM-Tastaturen mit 101 oder 102 Tasten gelten folgende Tasten als zusätzliche: die rechte Alt - und Ctrl -Taste im Hauptbereich der Tastatur, die Tasten Ins , Del , Home , End , PgUp , PgDown und die Cursor- Tasten im Block zwischen dem Nummernblock und der alphanumerischen Tastatur sowie die Divisions- und die Eingabetaste im Nummernblock. Auch andere Tastaturen unterstützen möglicherweise das Zusatztasten-Bit im Parameter *lParamHi*.

wm_SysKeyUp

Beschreibung Diese Botschaft wird übergeben, wenn der Anwender eine Taste losläßt, die gleichzeitig mit der Alt -Taste gedrückt wurde. Sie tritt auch auf, wenn kein Fenster den aktuellen Fokus hat. In diesem Fall wird die Botschaft *wm_SysKeyDown* an das aktive Fenster übergeben. Das empfangende Fenster kann zwischen den beiden Kontexten durch Überprüfung des Kontextcodes im Parameter *lParamHi* unterscheiden.

Parameter *wParam:* Enthält den virtuellen Tastencode der losgelassenen Taste.

lParamLo: Wiederholungszähler. Der Wert ergibt sich aus der Zeit, die der Anwender die Taste gedrückt hält und entspricht der ermittelten Anzahl von Tastenwiederholungen.

lParamHi: Die Bits von *lParamHi* können folgende Werte annehmen:

Bit	Bedeutung
0 bis 7	Enthalten den Scan-Code der Taste (OEM-abhängiger Wert).
8	Ist 1, wenn die gedrückte Taste eine zusätzliche Taste (zum Beispiel Funktionstaste oder Taste des numerischen Tastenblocks) ist.
9 bis 12	Werden entweder überhaupt nicht oder nur intern von Windows verwendet.
13	Ist 1, wenn gleichzeitig mit der Taste die Alt -Taste gedrückt wurde.
14	Ist 1, wenn die Taste vor Übergabe dieser Botschaft bereits gedrückt war.
15	Ist 1, wenn die Taste losgelassen, 0, wenn sie gedrückt wurde.

Rückgaberwert Wird nicht verwendet.

Hinweis Ist der Kontextcode 0, kann die Botschaft an die Funktion *TranslateAccelerator* weitergegeben werden.

Diese behandelt die Botschaft wie eine normale Tastaturbotschaft, nicht wie eine Systemtastenbotschaft. Dies ermöglicht es, Accelerator-Tasten zusammen mit dem aktiven Fenster zu verwenden, auch dann, wenn dieses nicht den Fokus hat.

Bei erweiterten IBM-Tastaturen mit 101 oder 102 Tasten gelten folgende Tasten als zusätzliche: die rechte Alt - und Ctrl -Taste im Hauptbereich der Tastatur, die Tasten Ins , Del , Home , End , PgUp , PgDown und die Cursor-Tasten im Block zwischen dem Nummernblock und der alphanumerischen Tastatur sowie die Divisions- und die Eingabetaste im Nummernblock.

Auch andere Tastaturen unterstützen möglicherweise das Zusatztasten-Bit im Parameter *lParamHi*.

Bei anderen erweiterten Tastaturen mit 102 Tasten wird die rechte Alt -Taste wie eine Ctrl -Alt -Tastenkombination behandelt.

Die folgende Tabelle zeigt die Folge der Botschaften, die sich aus dem Drücken und Loslassen dieser Taste ergeben:

Reihenfolge	Botschaft	Virtueller Tastencode (*lParamHi*)
1	*wm_KeyDown*	*vk_Control*
2	*wm_KeyDown*	*vk_Menu*
3	*wm_KeyUp*	*vk_Control*
4	*wm_SysKeyUp*	*vk_Menu*

wm_TimeChange

Beschreibung Diese Botschaft tritt auf, wenn eine Anwendung die Systemzeit (auch mehrfach) verändert. Alle Anwendungen, die dies tun, sollten diese Botschaft an alle Hauptfenster übergeben.

Parameter *wParam:* Wird nicht verwendet.

lParam: Wird nicht verwendet.

Rückgabewert Wird nicht verwendet.

Hinweis Die Anwendung kann die Funktion *SendMessage* verwenden, um die Botschaft *wm_TimeChange* an alle Hauptfenster zu übergeben. Dabei muß der Parameter *Wnd* auf $FFFF gesetzt sein.

wm_Timer

Beschreibung Diese Botschaft tritt auf, wenn die Zeitbegrenzung des vorgegebenen Timers überschritten wurde.

Parameter *wParam:* Enthält die Timer-ID, ein Integer, der den Timer bezeichnet.

lParam: Zeigt auf eine Funktion, die bei der Erstellung des Timers an die Funktion *SetTimer* übergeben wurde. Windows ruft die festgelegte Funktion direkt auf, wenn der Parameter *lParam* den Wert 0 hat. Andernfalls wird die Botschaft *wm_Timer* an die Fensterfunktion übergeben.

Rückgabewert Wird nicht verwendet.

wm_Undo

Beschreibung Diese Botschaft macht die vorhergehende Operation rückgängig. Nach Übergabe der Botschaft an ein Editierfeld wird der gerade gelöschte Text wiederhergestellt bzw. der zuletzt hinzugefügte Text gelöscht.

Parameter *wParam:* Wird nicht verwendet.

lParam: Wird nicht verwendet.

Rückgabewert Wird nicht verwendet.

wm_VKeyToItem

Beschreibung Diese Botschaft wird als Reaktion auf eine *wm_KeyDown*-Botschaft von einer Liste des Stils *lbs_WantKeyboardInput* an seinen Besitzer übergeben.

Parameter *wParam:* Enthält den virtuellen Tastencode der vom Anwender gedrückten Taste.

lParamLo: Das Handle der Liste.

lParamHi: Die aktuelle Position des Caret.

Rückgabewert Der Rückgabewert bestimmt die Aktion, die von der Anwendung als Reaktion auf diese Botschaft ausgeführt wurde.

Ein Rückgabewert von -2 zeigt an, daß die Anwendung alle Aspekte der Auswahl eines Eintrags bearbeitet hat und keine weiteren Aktionen in der Liste gewünscht werden. Der Rückgabewert -1 zeigt an, daß die Liste die Standardaktion als Reaktion auf den Tastendruck durchführen sollte. Ist der Rückgabewert 0 oder größer, bestimmt er den Index des Eintrags in der Liste und zeigt an, daß die Liste die Standardaktion durchführen soll, die dem Tastendruck und dem vorgegebenen Eintrag entspricht.

wm_VScroll

Beschreibung Diese Botschaft wird übergeben, wenn der Anwender die vertikale Bildlaufleiste anklickt.

Parameter *wParam:* Enthält einen Bildlaufleistencode, der die Bildlaufwünsche des Anwenders bestimmt. Folgende Werte sind möglich:

Wert	Bedeutung
sb_Bottom	Bildlauf nach unten.
sb_EndScroll	Ende des Bildlaufs.
sb_LineDown	Bildlauf eine Zeile nach unten.
sb_LineUp	Bildlauf eine Zeile nach oben.
sb_PageDown	Bildlauf eine Seite nach unten.
sb_PageUp	Bildlauf eine Seite nach oben.
sb_ThumbPosition	Bildlauf zu einer absoluten Position.
sb_ThumbTrack	Mauszug zur festgelegten Position.
sb_Top	Bildlauf nach oben.

lParamLo: Wird nicht verwendet.

lParamHi: Enthält das Handle der Bildlaufleiste. *lParamHi* ist 0, wenn sie mit dem Stil *ws_VScroll* erzeugt wurde.

Rückgabewert Wird nicht verwendet.

Hinweis Die Botschaft *sb_ThumbTrack* wird üblicherweise von Anwendungen benutzt, die auf Mauszüge reagieren.

Führt eine Anwendung einen Bildlauf in einem Fenster durch, muß sie die Mauszugposition durch die Funktion *SetScrollPos* zurücksetzen.

wm_VScrollClipboard

Beschreibung Diese Botschaft wird übergeben, wenn die Zwischenablage ein Daten-Handle des Formats *cf_OwnerDisplay* (d.h. der Besitzer der Zwischenablage sollte den Inhalt der Zwischenablage darstellen) enthält und ein Ereignis in der vertikalen Bildlaufleiste der Zwischenablage auftritt.

Parameter *wParam:* Enthält das Handle des Fensters der Zwischenablage.

lParamLo: Enthält einen der folgenden Bildlaufleistencodes:

Wert	Bedeutung
sb_Bottom	Bildlauf nach unten rechts.
sb_EndScroll	Ende des Bildlaufs.
sb_LineDown	Bildlauf eine Zeile nach unten.
sb_LineUp	Bildlauf eine Zeile nach oben.
sb_PageDown	Bildlauf eine Seite nach unten.
sb_PageUp	Bildlauf eine Seite nach oben.
sb_ThumbPosition	Bildlauf zu einer absoluten Position.
sb_Top	Bildlauf nach oben links.

lParamHi: Wird nicht verwendet.

Rückgabewert Wird nicht verwendet.

Hinweis Der Besitzer der Zwischenablage sollte die Funktion *InvalidateRect* verwenden oder die Zwischenablage wie gewünscht neu darstellen. Die Bildlaufleistenposition sollte mit Hilfe der Funktion *SetScrollPos* zurückgesetzt werden. Daten und Format der Zwischenablage werden mit der Funktion *SetClipboardData* gesetzt.

wm_WinIniChange

Beschreibung Diese Botschaft wird übergeben, wenn die Initialisierungsdatei WIN.INI verändert wird. Ändert eine Anwendung die Datei WIN.INI, sollte sie diese Botschaft an alle Hauptfenster übergeben.

Parameter *wParam:* Wird nicht verwendet.

lParam: Zeigt auf einen String, der den Namen des geänderten Abschnitts angibt. Die eckigen Klammern sind in diesen String nicht eingeschlossen.

Rückgabewert Wird nicht verwendet.

Hinweis Die Anwendung kann die Funktion *SendMessage* verwenden, um die Botschaft an alle Hauptfenster zu übergeben. Dabei muß der Parameter *Wnd* auf $FFFF gesetzt sein.

Auch wenn es eigentlich falsch ist, übergeben einige Anwendungen diese Botschaft, wobei der Parameter *lParam* auf **nil** gesetzt ist. Empfängt eine Anwendung eine derartige Botschaft, sollte sie alle Abschnitte der Datei WIN.INI überprüfen, die sie selbst betreffen könnten.

4. Windows-Typen

Windows definiert eine Anzahl von Typen und Strukturen. ObjectWindows stellt dafür Turbo-Pascal-Äquivalente zur Verfügung, die in der Unit *WinTypes* definiert und in diesem Kapitel dokumentiert sind.

Bool-Typ

WinTypes

Deklaration
```
Bool = System.WordBool;
```

Beschreibung *Bool* entspricht exakt dem Turbo-Pascal-Standardtyp *WordBool* und ist dafür vorgesehen, Kompatibilität mit Windows-Code, der in anderen Sprachen geschrieben wurde, herzustellen.

HBitmap-Typ

WinTypes

Deklaration
```
HBitmap = THandle;
```

Beschreibung *HBitmap* ist ein Handle-Typ für Bitmap-Handles.

HBrush-Typ

WinTypes

Deklaration
```
HBrush = THandle;
```

Beschreibung Definiert einen Handle-Typ für Pinselzeichenwerkzeuge.

HCursor-Typ

WinTypes

Deklaration
```
HCursor = THandle;
```

Beschreibung *HCursor* definiert einen Handle-Typ für Mauszeiger-Handles.

HDC-Typ

WinTypes

Deklaration
```
HDC = THandle;
```

Beschreibung *HDC* definiert einen Handle-Typ für Gerätekontext-Handles. Auch Bildschirmkontexte sind Gerätekontexte, so daß deren

Handles immer in Variablen des Typs *HDC* gespeichert werden.

HFont-Typ **WinTypes**

Deklaration `HFont = THandle;`

Beschreibung *HFont* definiert einen Handle-Typ für Schriftzeichenwerkzeuge.

HIcon-Typ **WinTypes**

Deklaration `HIcon = THandle;`

Beschreibung *HIcon* definiert einen Handle-Typ für Symbol-Handles.

HMenu-Typ **WinTypes**

Deklaration `HMenu = THandle;`

Beschreibung *HMenu* definiert einen Handle-Typ für Menü-Ressourcen.

HPalette-Typ **WinTypes**

Deklaration `HPalette = THandle;`

Beschreibung *HPalette* definiert einen Handle-Typ für Paletten-Handles.

HPen-Typ **WinTypes**

Deklaration `HPen = THandle;`

Beschreibung *HPen* definiert einen Handle-Typ für Stiftzeichenwerkzeuge.

HRgn-Typ **WinTypes**

Deklaration `HRgn = THandle;`

Beschreibung *HRgn* definiert einen Handle-Typ für Region-Handles.

HStr-Typ **WinTypes**

Deklaration `HStr = THandle;`

Beschreibung *HStr* definiert einen Handle-Typ für String-Handles.

HWnd-Typ **WinTypes**

Deklaration
```
HWnd = THandle;
```

Beschreibung *HWnd* definiert einen Handle-Typ für Fenster-Handles. Diese werden im allgemeinen von ObjectWindows-Schnittstellenobjekten benutzt, um ihre zugeordneten Windows-Schnittstellenelemente zu verwalten. Viele API-Funktionsaufrufe erfordern ein Fenster-Handle, um anzuzeigen, welches Fenster sie bearbeiten sollen.

LPHandle-Typ **WinTypes**

Deklaration
```
LPHandle = PHandle;
```

Beschreibung *LPHandle* definiert einen Zeiger auf ein Handle. Er wird von ObjectWindows nicht benutzt, aber er wurde aufgenommen, um Kompatibilität mit Windows-Code herzustellen, der in anderen Sprachen geschrieben wurde.

LPVoid-Typ **WinTypes**

Deklaration
```
LPVoid = Pointer;
```

Beschreibung *LPVoid* definiert einen Zeiger. Er wird von ObjectWindows nicht verwendet, aber er wurde aufgenommen, um Kompatibilität mit Windows-Code herzustellen, der in anderen Sprachen geschrieben wurde.

MakeIntAtomType **WinTypes**

Deklaration
```
MakeIntAtom = PStr;
```

Beschreibung *MakeIntAtom* wird bei der Typumwandlung von Integerzahlen in Atome verwendet.

Dies ist gleichbedeutend mit der Typumwandlung in den Turbo-Pascal-Typ *PChar*.

MakeIntResource-Typ **WinTypes**

Deklaration
```
MakeIntResource = PStr;
```

Beschreibung *MakeIntResource* wird bei der Typumwandlung von Integerzahlen in Ressourcenbezeichnungen verwendet.

Dies ist gleichbedeutend mit der Typumwandlung in den Turbo-Pascal-Typ *PChar*.

PBool-Typ **WinTypes**

Deklaration `PBool = ^WordBool;`

Beschreibung *PBool* definiert einen Zeiger auf einen 16-Bit-Boolean-Wert.

PByte-Typ **WinTypes**

Deklaration `PByte = ^Byte;`

Beschreibung *PByte* definiert einen Zeiger auf einen vorzeichenlosen 8-Bit-Wert.

PHandle-Typ **WinTypes**

Deklaration `PHandle = ^THandle;`

Beschreibung *PHandle* definiert einen Zeiger auf ein generisches Windows-Handle.

PInteger-Typ **WinTypes**

Deklaration `PInteger = ^Integer;`

Beschreibung *PInteger* definiert einen Zeiger auf eine vorzeichenbehaftete 16-Bit-Zahl.

PLongInt-Typ **WinTypes**

Deklaration `PLongInt = ^LongInt;`

Beschreibung *PLongInt* definiert einen Zeiger auf eine 32-Bit-Integerzahl.

PStr-Typ **WinTypes**

Deklaration `PStr = PChar;`

Beschreibung *PStr* definiert einen Zeiger auf einen nullterminierten String. Er ist äquivalent zum Turbo-Pascal-Typ *PChar* und dafür vorgesehen, Kompatibilität mit Code herzustellen, der in anderen Sprachen geschrieben wurde.

PWord-Typ **WinTypes**

Deklaration `PWord = ^Word;`

Beschreibung *PWord* definiert einen Zeiger auf eine vorzeichenlose 16-Bit-Zahl.

TAtom-Typ WinTypes

Deklaration
```
TAtom = Word;
```

Beschreibung *TAtom* definiert eine 16-Bit-Zahl, die ein Atom oder eine Botschaft bezeichnet, die zwischen DDE betreibenden Anwendungen ausgetauscht werden soll.

TBitmap-Typ WinTypes

Deklaration
```
TBitmap = record
  bmType: Integer;
  bmWidth: Integer;
  bmHeight: Integer;
  bmWidthBytes: Integer;
  bmPlanes: Byte;
  bmBitsPixel: Byte;
  bmBits: Pointer;
end;
```

Beschreibung *TBitmap*-Records werden von den Funktionen *CreateBitmapIndirect* und *GetObject* benutzt, um die Größe, die Farben und die Bitwerte für ein Bitmap zu beschreiben.

Das Feld *bmType* definiert den Bitmap-Typ. Ein Wert von 0 kennzeichet ein virtuelles Bitmap. *bmWidth* und *bmHeight* definieren die Breite in Pixel bzw. die Höhe in Rasterzeilen des Bitmaps. Beide müssen größer als 0 sein. *bmWidthBytes* gibt die Anzahl von Bytes in jeder Rasterzeile an und muß eine gerade Zahl sein.

bmPlanes und *bmBitsPixel* geben die Anzahl der Farbebenen bzw. die Anzahl der benachbarten Farbbits auf jeder Ebene an.

bmBits ist ein Zeiger auf die eigentlichen Bits, die das Bitmap bilden. Die Bits haben die Form eines Arrays von Bytes.

TBitmapCoreHeader-Typ WinTypes

Deklaration
```
TBitmapCoreHeader = record
  bcSize: LongInt;
  bcWidth: Word;
  bcHeight: Word;
  bcPlanes: Word;
  bcBitCount: Word;
end;
```

Beschreibung *TBitmapCoreHeader* definiert die Größe und die Farben eines geräteunabhängigen Bitmaps. Die Records *TBitmapCoreHeader* werden als Teil von *TBitmapCoreInfo*-Records verwendet, um geräteunabhängige Bitmaps vollständig zu definieren.

bcSize ist die Anzahl der Bytes im Record *TBitmapCoreHeader*.

bcWidth und *bcHeight* sind die Breite, bzw. die Höhe (in Pixel) des Bitmaps. *bcPlanes* gibt die Anzahl von Ebenen für das Zielgerät an; sie muß auf 1 gesetzt werden. *bcBitCount* gibt die Anzahl von Bits pro Pixel an. Zulässige Werte für *bcBitCount* sind 1, 2, 8 und 24.

Die Werte von *bcBitCount* haben folgende Bedeutung:

- Wenn *bcBitCount* 1 ist, ist das Bitmap monochrom, die Farbtabelle muß zwei Einträge enthalten und jedes Bit im Bitmap steht für ein Pixel. Ein gelöschtes Bit repräsentiert die erste Farbe in der Tabelle, ein gesetztes Bit die zweite.

- Wenn *bcBitCount* 4 ist, hat das Bitmap bis zu 16 Farben, die von 0 bis 15 numeriert sind, so daß jedes Pixel im Bitmap vier Bits zur Kennzeichnung seiner Farbe beansprucht. Die Farbtabelle enthält 16 Einträge. Folglich repräsentiert jedes Byte im Bitmap zwei Pixel, zuerst das höherwertige und dann das niederwertige Halb-Byte.

- Wenn *bcBitCount* 8 ist, hat das Bitmap bis zu 256 Farben, so daß ein ganzes Byte benötigt wird, um jedes Pixel zu repräsentieren. Jedes Byte im Bitmap steht damit für einen Index in die Farbtabelle zwischen 0 und 255.

- Wenn *bcBitCount* 24 ist, hat das Bitmap bis zu 2^{24} Farben. Es gibt keine Farbtabelle und jedes Pixel wird von drei Bytes repräsentiert, die die Intensitäten des roten, grünen und gelben Farbanteils im Pixel bestimmen.

TBitmapCoreInfo-Typ

WinTypes

Deklaration

```
TBitmapCoreInfo = record
  bmciHeader: TBitmapCoreHeader;
  bmciColors:    array [0..0]  of TRGBTriple;
end;
```

Beschreibung

TBitmapCoreInfo-Records kombinieren Information über die Größe und die Farbe aus einem *TBitmapCoreHeader*-Record mit einer Farbinformationstabelle, um ein geräteunabhängiges Bitmap vollständig zu definieren.

bmciHeader ist ein *TBitmapCoreHeader*-Record, der die Grössen- und Farbinformation für das Bitmap definiert. Das Feld *bcSize* von *bmciHeader* kann verwendet werden, um einen Offset für das Feld *bmciColors* anzugeben.

bmciColors ist ein Array von *TRGBTriple*-Records. Die Anzahl von Elementen in diesem Array wird vom Feld *bcBitCount* des Feldes *bmciHeader* festgelegt. *bmciColors* kann entweder ein Array von RGB-Farb-Records sein oder ein Array von Indizes in die aktuell realisierte virtuelle Palette. Die Art, in der das Feld

interpretiert wird, hängt vom Parameter *Usage* ab, der an die Funktion übergeben wird, die auf das geräteunabhängige Bitmap zugreift. In Kapitel 1 »Windows-Stile und -Konstanten« finden sich im Abschnitt »DIB_Xxxx« weitere Informationen.

Der Einsatz von RGB-Werten für das Feld *bmciColors* wird sehr empfohlen, wenn das Bitmap nicht ausschließlich von einer einzigen Anwendung benutzt wird. Wenn das Bitmap in einer Datei gespeichert wird oder an ein anderes Programm übertragen wird, sollte keine Palettenindizierung verwendet werden.

TBitmapFileHeader-Typ

WinTypes

Deklaration

```
TBitmapFileHeader = record
  bfType: Word;
  bfSize: LongInt;
  bfReserved1: Word;
  bfReserved2: Word;
  bfOffBits: LongInt;
end;
```

Beschreibung

TBitmapFileHeader-Records definieren den Header einer geräteunabhängigen Bitmap-Datei. Sie enthalten Daten, die die Typgröße und das Layout der Bitmap-Datei definieren.

Eine geräteunabhängige Bitmap-Datei besteht aus einem Record *TBitmapFileHeader*, gefolgt von einem der beiden Records *TBitmapInfo* oder *TBitmapCoreInfo* und anschließend den eigentlichen Bitmap-Daten. Die Felder im Bitmap-Datei-Header werden wie folgt definiert:

- Das Feld *bfType* gibt den Typ der Datei an. Er muß BM sein.
- Das Feld *bfSize* gibt die Größe der Datei in 4-Byte-Blöcken an.

bfReserved1 und *bfReserved2* sind von Windows reserviert.

Das Feld *bfOffBits* sagt aus, nach wievielen Bytes ab Dateianfang die eigentliche Bitmap-Information beginnt.

TBitmapInfo-Typ

WinTypes

Deklaration

```
TBitmapInfo = record
  bmiHeader: TBitmapInfoHeader;
  bmiColors:   array [0..0]  of TRGBQuad;
end;
```

Beschreibung

TBitmapInfo-Records enthalten Größen- und Farbinformationen für geräteunabhängige Bitmaps für Windows 3.0. Das eigentliche Bitmap ist definiert als ein Array von Bytes, das die Pixel des Bitmaps repräsentiert.

Das Feld *bmiHeader* enthält einen Record *TBitmapInfo-Header*, der die Größen- und Farbformatierung für das Bitmap definiert. Das Feld *bmiColors* ist ein Array von *TRGBQuad*-Records, die die Farben des Bitmaps definieren. Die Anzahl der Einträge im Array wird vom Wert des Feldes *biBitCount* des Records *bmiHeader* festgelegt.

TBitmapInfoHeader-Typ

WinTypes

Deklaration

```
TBitmapInfoHeader = record
  biSize: LongInt;
  biWidth: LongInt;
  biHeight: LongInt;
  biPlanes: Word;
  biBitCount: Word;
  biCompression: LongInt;
  biSizeImage: LongInt;
  biXPelsPerMeter: LongInt;
  biYPelsPerMeter: LongInt;
  biClrUsed: LongInt;
  biClrImportant: LongInt;
end;
```

Beschreibung

TBitmapInfoHeader-Records werden von *TBitmapInfo*-Records benutzt, um die Abmessungs- und Farbformatierung eines geräteunabhängigen Bitmaps für Windows 3.0 zu definieren.

Das Feld *biSize* gibt die Größe des Records in Byte an. *biWidth* und *biHeight* sind jeweils Breite und Höhe des Bitmaps in Pixel. *biPlanes* ist die Anzahl der Ebenen für das Zielgerät; diese muß auf 1 gesetzt sein. *biBitCount* gibt die Anzahl der Bits an, die erforderlich sind, um jedes Pixel im Bitmap zu beschreiben. *biCompression* gibt den Typ der Kompression an, die für das Bitmap benutzt werden soll; es kann eine der *bi_*-Konstanten sein, die in Kapitel 1 »Windows-Stile und -Konstanten« im Abschnitt »bi_Xxxx« definiert sind. *biSizeImage* ist die Größe des Bitmap-Bildes in Byte. *biXPelsPerMeter* und *biYPelsPerMeter* geben die horizontale bzw. die vertikale Auflösung des Zielgerätes für das Bitmap an.

Das Feld *biClrUsed* wird verwendet, um die Anzahl der Farbtabelleneinträge anzugeben, die vom Bitmap tatsächlich benutzt werden. Der Wert *biBitCount* bestimmt die Maximalzahl der Einträge: Eine 0 in *biClrUsed* zeigt an, daß die Maximalzahl verwendet wurde, ein Wert zwischen 1 und 23 zeigt an, auf wieviele Farben tatsächlich zugegriffen wurde. Wenn *biBitCount* 24 ist, ist *biClrUsed* die Größe der Referenzfarbtabelle, die von Windows benutzt wurde, um die Leistungsfähigkeit der Farbpalette zu optimieren.

Das Feld *biClrImportant* gibt die Anzahl der Farben an, die

für die Darstellung des Bitmaps als wichtig erachtet wurden. Ein Wert von 0 zeigt an, daß alle Farben wichtig sind.

Die Bedeutungen der Werte von *biBitCount* entsprechen exakt denen, die für das Feld *bcBitCount* Pixeln der *TBitCoreHeader*-Records definiert sind.

TCatchBuf-Typ

WinTypes

Deklaration

```
TCatchBuf = array[0..8] of Integer;
```

Beschreibung *TCatchBuf* ist ein Record, der den Status aller Systemregister enthält. Er wird von den Routinen *Catch* und *Throw* benutzt.

TClientCreateStruct-Typ

WinTypes

Deklaration

```
TClientCreateStruct = record
  hWindowMenu: THandle;
  idFirstChild: Word;
end;
```

Beschreibung *TClientCreateStruct* wird benutzt, wenn MDI-Client-Fenster erzeugt werden, um Information über die Fenster-ID und das Menü zu enthalten. Das Feld *hWindowMenu* ist ein Handle zum Menü der Anwendung. *idFirstChild* ist die untergeordnete Fenster-ID des ersten untergeordneten Fensters der MDI-Anwendung. Die IDs untergeordneter Fenster werden von Windows eindeutig zugewiesen und verwaltet.

TColorRef-Typ

WinTypes

Deklaration

```
TColorRef = LongInt;
```

Beschreibung *TColorRef* ist ein 32-Bit-Wert, der einer Farbe entspricht und von zahlreichen GDI-Funktionen verwendet wird. Er kann, abhängig vom Wert des höherwertigen Bytes des höherwertigen Word, im LongInt auf drei verschiede Arten interpretiert werden.

Wenn das höchstwertige Byte 0 ist, stehen die nächsten drei Bytes für RGB-Farbintensitäten (jeweils für Blau, Grün und Rot). Damit repräsentiert der Wert $00FF0000 reines Blau mit voller Intensität, $0000FF00 reines Grün, $000000FF reines Rot, $00000000 Schwarz und $00FFFFFF Weiß. RGB-Werte können am einfachsten mit der Funktion *RGB* in *TColorRef*-Werte konvertiert werden.

Wenn das höchstwertige Byte 1 ist, muß das nächste Byte 0 sein. Das niederwertige Word (die nächsten beiden Bytes) bilden einen Index in eine virtuelle Palette.

Somit ist $01000000 der Index 0 (der erste Eintrag) einer

Palette. Palettenindizes vom Typ Integer können mit der Funktion *PaletteIndex* in *TColorRef*-Werte konvertiert werden.

Wenn das höchstwertige Byte 2 ist, repräsentieren die nächsten drei Bytes RGB-Farbintensitäten (wie mit RGB-Werten, die für ein höchstwertiges Byte von 0 beschrieben wurden). Der Wert wird jedoch hier an die ähnlichste Farbe in der virtuellen Palette im aktuellen Gerätekontext angeglichen. Palettenbezogene RGB-*TColorRef*-Werte können mit der Funktion *PaletteRGB* aus RGB-Werten ermittelt werden.

Um zu erreichen, daß die palettenindex- oder palettenbezogenen *TColorRef*-Werte mit einem Gerätekontext arbeiten, muß eine Anwendung mit eigener Palette mit *SelectPalette* ihre Palette in dem Gerätekontext auswählen, für den sie geschrieben ist, und danach mit *RealizePalette* realisieren. Damit verfügt sie über Zeichenfunktionen, die die richtigen Farben aus der Palette benutzen. Analog dazu muß die Palette vor dem Erzeugen eines virtuellen Zeichenwerkzeugs ausgewählt und realisiert werden, damit die Farben gültig werden.

TCompareItemStruct-Typ

WinTypes

deklaration

```
TCompareItemStruct = record
  CtlType: Word;
  CtlID: Word
  hwndItem: HWnd;
  itemID1: Word;
  itemData1: LongInt;
  itemID2: Word;
  itemData2: LongInt;
end;
```

Beschreibung

TCompareItemStruct-Records werden dazu verwendet, Einträge in sortierten, benutzerdefinierten Kombinationsfenstern oder Listenfenstern zu vergleichen. Das Hinzufügen von Einträgen zu solchen Steuerelementen erzeugt eine Botschaft *wm_CompareItem*, einer der Parameter dieser Botschaft ist ein Zeiger auf einen *TCompareItemStruct*-Record. Nach dem Empfang dieser Botschaft vergleicht der Besitzer die Einträge im Record und gibt ergebnisabhängige Werte zurück. Siehe auch die Beschreibung der *wm_CompareItem*-Botschaft in Kapitel 3 »Windows-Botschaften«.

Das Feld *CtlType* enthält eine *odt_*-Konstante, die anzeigt, daß das Steuerelement ein benutzerdefiniertes Kombinationsfenster *(odt_ComboBox*) oder eine Liste (*odt_ListBox*) ist. *CtlID* und *hwndItem* sind die Steuerelement-ID bzw. das Fenster-Handle für das Steuerelement.

itemID1 und *itemData1* geben den Index des ersten Eintrags im Listen- oder Kombinationsfenster, der verglichen wird,

sowie die Daten für diesen Eintrag an. Die Daten von *itemData1* entsprechen denen des Parameters *lParam* der Botschaft, die den Eintrag zur Liste hinzufügte. *ItemID2* und *itemData2* bieten in analoger Weise dieselbe Information für den zweiten Eintrag.

TComStat-Typ

WinTypes

Deklaration

```
TComStat = record
  Flags: Byte;
  cbInQe: Word;
  cbOutQue: Word;
end;
```

Beschreibung

TComStat-Records beinhalten die Statusinformation über eine Schnittstelle. Sie werden von der Funktion *GetCommError* verwendet.

Das Feld *Flags* ist ein Bit-Feld mit den Bits, die von den *com_*-Konstanten definiert sind.

Die Felder *cbInQue* und *cbOutQue* geben die Anzahl der Zeichen in der Empfangs- bzw. Sende-Warteschlange an.

TCreateStruct-Typ

WinTypes

Deklaration

```
TCreateStruct = record
  lpCreateParams: PChar;
  hInstance: THandle;
  hMenu: THandle;
  hwndParent: HWnd;
  cy: Integer;
  cx: Integer;
  y: Integer;
  x: Integer;
  style: LongInt;
  lpszName: PChar;
  lpszClass: PChar;
  dwExStyle: LongInt;
end;
```

Beschreibung

TCreateStruct-Records werden benutzt, um Initialisierungsparameter an Fensterfunktionen einer Anwendung zu übergeben.

Das Feld *lpCreateParams* zeigt auf Erstellungsdaten des Fensters. *hInstance*, *hMenu* und *hwndParent* sind Handles zur Modulinstanz des zu dem Fenster gehörigen Moduls, zum Menü für das Fenster und zum übergeordneten Fenster des neuen Fensters. *hwndParent* ist 0, wenn das erzeugte Fenster das Hauptfenster der Anwendung ist.

cx und *cy* geben die Breite und die Höhe des Fensters und *x* und *y* die vertikalen und horizontalen Koordinaten der oberen linken Ecke des Fensters an. Die Koordinaten sind relativ zu einem übergeordneten Fenster, wenn ein solches vorhanden ist.

style enthält die Stil-Flags für das Fenster, *lpszName* und *lpszClass* zeigen auf die nullterminierten Strings, die die Bezeichnung bzw. Klassenbezeichnung des Fensters angeben. *dwExStyle* enthält erweiterte Stilinformation für das Fenster.

TDCB-Typ

WinTypes

Deklaration

```
TDCB = record;
  Id: Byte;
  BaudRate: Word;
  ByteSize: Byte;
  Parity: Byte;
  StopBits: Byte;
  RlsTimeout: Word;
  CtsTimeout: Word;
  DsrTimeout: Word;
  Flags: Word;
  XonChar: Char;
  XoffChar: Char;
  XonLim: Word;
  XoffLim: Word;
  PeChar: Char;
  EofChar: Char;
  EvtChar: Char;
  TxDelay: Word;
end;
```

Beschreibung

TDCB-Records enthalten Steuerelementinformation für diejenigen Schnittstellen, die von den Funktionen *BuildCommDCB, GetCommState* und *SetCommState* verwendet werden.

Das Feld *Id* ist die ID der Schnittstelle. Wenn das höherwertige Bit gesetzt ist (vgl. die Maske *LPTx*), handelt es sich um einen parallelen, andernfalls um einen seriellen Anschluß.

BaudRate, ByteSize, Parity und *StopBits* definieren die Übertragungsparameter der Schnittstelle. *ByteSize* ist die Anzahl der Bits in jedem Zeichen, der Wert kann zwischen 4 und 8 liegen. Für *Parity* wird eine der Datenübertragungskonstanten *EvenParity, MarkParity, NoParity, OddParity* oder *SpaceParity* eingesetzt. *StopBits* ist eine der Datenübertragungskonstanten *OneStopBit, One5StopBits,* oder *TwoStopBits*. Die Datenübertragungskonstanten werden in Kapitel 1 »Windows-Stile und -Konstanten« beschrieben.

RlsTimeout, CtsTimeout und *DsrTimeout* geben die Zeit, die

die Schnittstelle warten sollte, bevor sie das Warten auf die Signale RLSD, CTS bzw. DSR abbricht, in Millisekunden an.

fBinary legt fest, ob der Anschluß im Binärmodus arbeitet. Im Binärmodus fließen alle Daten unverändert durch, im Nicht-Binärmodus wird das Zeichen Eof (*EofChar*) verwendet, um das Datenende zu kennzeichnen.

Im Feld *Flags* repräsentiert jedes Bit einen Wechselschalter für eine andere Art der Fehlerprüfung. Die Bits werden in der folgenden Tabelle definiert. Auf einzelne Bits kann mittels der *dcb_*-Konstanten zugegriffen werden.

Bit	Bedeutung, wenn gesetzt
fRtsDisable	RTS gesperrt
fParity	Paritätsprüfung eingeschaltet
fOutxCtsFlow	CTS beim Senden überwacht
fOutxDsrFlow	DSR beim Senden überwacht
fDummy	Reserviert
fDtrDisable	DTR gesperrt
fOutX	Xon/Xoff-Steuerung beim Senden benutzt
fInX	Xon/Xoff-Steuerung beim Empfangen benutzt
fPeChar	Paritätsfehler werden ersetzt
fNull	Nullzeichen werden verworfen
fChEvt	Das Zeichen *EvtChar* wird als Ereignis gekennzeichnet
fDtrFlow	DTR wird beim Empfangen als Flußsteuerung benutzt
fRtsFlow	RTS wird beim Empfangen als Flußsteuerung benutzt
fDummy2	Reserviert

XonChar und *XoffChar* geben jeweils die Werte der Zeichen Xon und Xoff für Senden bzw. Empfangen an. *XonLim* ist die Anzahl der Zeichen in der Empfangs-Warteschlange, die das Senden eines Xon, *XoffLim* die Anzahl der Zeichen, die das Senden eines Xoff auslöst.

PeChar, EofChar und *EvtChar* definieren die Zeichen, die jeweils dazu benutzt werden, um Paritätsfehler zu ersetzen, das Ende der Daten zu signalisieren und ein Ereignis anzuzeigen.

Das Feld *TxDelay* wird gegenwärtig nicht benutzt.

TDDEAck-Typ

WinTypes

Deklaration

```
TDDEAck = record
  Flags: Word;
end;
```

Beschreibung *TDDEAck*-Records enthalten Quittierungsinformationen, die in einem Parameter einer Botschaft *wm_dde_Ack* als Antwort

auf jede DDE-Botschaft – ausgenommen *wm_dde_Initiate* – gesendet wurden.

Das Feld *Flags* ist ein Bit-Feld vom Typ Word, von dem derzeit nur 2 Bits für den Gebrauch durch eine Anwendung definiert sind. Auf diese kann mit den *dde*-Konstanten zugegriffen werden. Das gesetzte Bit *fAck* zeigt an, daß die Anforderung akzeptiert wurde. Das gesetzte Bit *fBusy* zeigt an, daß die Anwendung nicht in der Lage ist, auf die Anforderung zu reagieren. *fBusy* hat nur dann eine Bedeutung, wenn *fAck* 0 ist.

Das niederwertige Byte von *Flags* ist das »Feld« *bAppReturnCode*, auf das mit der Maske *dde_AppReturnCode* zugegriffen werden kann. Es enthält anwendungsspezifische Rückgabecodes. Die verbleibenden Bits sind für Windows reserviert.

TDDEAdvise-Typ

WinTypes

Deklaration

```
TDDEAdvise = record
  Flags: Word;
  cfFormat: Integer
end;
```

Beschreibung

Der Record *TDDEAdvise* enthält eine Anforderung an einen DDE-Server und wird in einem Parameter einer Botschaft *wm_dde_Advise* übergeben.

Das Feld *Flags* ist ein Bit-Feld vom Typ Word mit nur 2 definierten Bits, *fAckReq* und *fDeferUpd*. Auf diese kann mit den *dde_*-Konstanten zugegriffen werden. Die 14 niederwertigen Bits von *Flags* sind nicht definiert, sie sind aber reserviert.

Wenn das Bit *fAckReq* gesetzt ist, wird der Server dazu aufgefordert, seine *wm_dde_Data*-Botschaften mit gesetztem Bit *fAckReq* zu senden, um die Botschaftsflußsteuerung zu implementieren. Wenn das Bit *fDeferUpd* gesetzt ist, wird der Server aufgefordert, *wm_dde_Data*-Botschaften mit auf 0 gesetztem Handle *hData* zu senden, um den Client auf geänderte Daten aufmerksam zu machen. Nach einer derartigen Benachrichtigung kann der Client eine Botschaft *wm_dde_-Request* senden, um die aktualisierten Daten anzufordern.

Das Feld *cfFormat* gibt das bevorzugte Datenformat des Client an, wobei eine der *cf_*-Zwischenablageformatkonstanten eingesetzt wird.

TDDEData-Typ

WinTypes

Deklaration

```
TDDEData = record
  Flags: Word;
  cfFormat: Integer;
  Value:   array [0..0]  of Char;
end;
```

Beschreibung *TDDEData*-Records enthalten Daten, die von einer Anwendung zur anderen übertragen werden. Sie werden als Parameter in den Botschaften *wm_dde_Data* übergeben.

Das Feld *Flags* ist ein Bit-Feld vom Typ Word mit gegenwärtig nur drei definierten Bits. Auf diese Bits, *fAckReq, fRelease* und *fRequested,* kann mit den *dde_*-Konstanten zugegriffen werden. Wenn das Bit *fAckReq* gesetzt ist, sollte die Anwendung den Empfang der Daten mit einer Botschaft *wm_dde_Ack* bestätigen. Bei gesetztem Bit *fRelease* sollte der Client die mit der Botschaft *wm_dde_Data* gesendeten Daten nach der Verarbeitung freigeben. Ist das Bit *fRequested* gesetzt, werden die zugehörigen Daten als Antwort auf eine Aufforderung vom Client übergeben. Alle anderen Bits von *Flags* sind reserviert.

Das Feld *cfFormat* enthält eine *cf_*Zwischenablageformatkonstante, die das Format anzeigt, in dem die Daten an den Client gesendet werden.

Das Feld *Value* enthält die Daten, die übertragen werden, in dem von *cfFormat* vorgegebenen Format.

TDDEPoke-Typ

WinTypes

Deklaration

```
TDDEPoke = record
  Flags: Word;
  cfFormat: Byte;
  Value:   array [0..0]  of Byte;
end;
```

Beschreibung *TDDEPoke*-Records enthalten nicht angeforderte Daten, die eine Botschaft *wm_dde_Poke* begleiten.

Das Feld *Flags* ist ein Bit-Feld vom Typ Word, wobei derzeit nur das Bit *fRelease* definiert ist. Dieses Bit zeigt an, daß der Empfänger die Daten nach der Verarbeitung freigeben sollte. Alle anderen Bits von *Flags* sind reserviert. Auf *fRelease* kann mit der Konstante *dde_Release* zugegriffen werden.

Das Feld *cfFormat* bezeichnet das bevorzugte Datenformat des Client, wobei eine der *cf_*-Zwischenablageformatkonstanten benutzt wird.

Das Feld *Value* enthält die Daten, die übertragen werden, in dem von *cfFormat* vorgegebenen Format.

TDeleteItemStruct-Typ

WinTypes

Deklaration

```
TDeleteItemStruct = record
  CtlType: Word;
  CtlID: Word;
  itemID: Word;
  hwndItem: HWnd;
```

```
  itemData: LongInt;
end;
```

Beschreibung *TDeleteItemStruct*-Records werden verwendet, um einen aus einem besitzergezeichneten Kombinations- oder Listenfenster gelöschten Eintrag zu beschreiben. Die Botschaft *wm_DeleteItem* gelangt zum Besitzer des Fensters, wobei der Parameter *lParam* auf einen *tDeleteItemStruct*-Record zeigt.

Das Feld *CtlType* kennzeichnet den Typ des Steuerelements: Liste (*odt_ListBox*) oder Kombinationsfenster (*odt_ComboBox*). *CtlId* ist die Steuerelement-ID des Fensters, *itemID* der Index des gelöschten Eintrags.

hwndItem ist das Fenster-Handle des Steuerelements und *itemData* der 32-Bit-Wert des indizierten Eintrags.

TDevMode-Typ

WinTypes

Deklaration

```
TDevMode = record
  dmDeviceName:    array [0..cchDeviceName-1]    of Char;
  dmSpecVersion: Word;
  dmDriverVersion: Word;
  dmSize: Word;
  dmDriverExtra: Word;
  dmFields: LongInt;
  dmOrientation: Integer;
  dmPaperSize: Integer;
  dmPaperLength: Integer;
  dmPaperWidth: Integer;
  dmScale: Integer;
  dmCopies: Integer;
  dmDefaultSource: Integer;
  dmPrintQuality: Integer;
  dmColor: Integer;
  dmDuplex: Integer;
end;
```

Beschreibung *TDevMode*-Records werden von den Routinen *DeviceCapabilities* und *ExtDeviceMode* verwendet, um Information über einen Druckertreiber zu speichern.

Das Feld *dmDeviceName* enthält einen nullterminierten String, der die Bezeichnung des unterstützten Gerätes angibt. *dmSpecVersion* ist die Versionsnummer der Datenspezifikation, gegenwärtig $0300.

dmDriverVersion ist die Versionsnummer des Treibers, die vom Entwickler vorgegeben wird.

dmSize ist die Größe des Records ohne Berücksichtigung des Feldes *dmDriverData* am Ende.

dmDriverExtra gibt die Größe des Feldes *dmDriverData* an.

dmFields ist ein Bit-Feld vom Typ LongInt, das anzeigt, welche der verbleibenden Felder initialisiert wurden. Jedes Bit gehört zu einem Feld:

Flag	Feld
dm_Color	*dmColor*
dm_Copies	*dmCopies*
dm_DefaultSource	*dmDefaultSource*
dm_Duplex	*dmDuplex*
dm_Orientation	*dmOrientation*
dm_PaperLength	*dmPaperLength*
dm_PaperSize	*dmPaperSize*
dm_PaperWidth	*dmPaperWidth*
dm_PrintQuality	*dmPrintQuality*
dm_Scale	*dmScale*
dm_SpecVersion	*dmSpecVersion*

Das Feld *dmOrientation* wählt die Papierausrichtung – Portrait oder Landscape – unter Verwendung der Konstanten *dmorient_* aus.

Das Feld *dmPaperSize* wählt die Papiergröße aus, wobei eine der *dmpaper*-Konstanten verwendet wird.

Mit den Feldern *dmPaperLength* und *dmPaperWidth* können die im Feld *dmPaperSize* angegebene Papierlänge und -breite überschrieben werden.

Das Feld *dmScale* skaliert die gedruckte Ausgabe um einen Faktor von *dmScale*/100. Beispielsweise reduziert ein Wert von 75 Bilder auf 75% ihrer normalen Größe.

Das Feld *dm_Copies* wählt die Anzahl der zu druckenden Kopien aus.

Das Feld *dmDefaultSource* kennzeichnet mit einer der *dmbin_*-Konstanten den Schacht, aus dem das Papier per Voreinstellung eingezogen wird.

dmPrintQuality gibt die zu verwendende Druckauflösung an. Es kann entweder eine der geräteunabhängigen *dmres_*-Konstanten (negative Werte) angegeben werden oder eine geräteabhängige positive Zahl, deren Wert die Punkte pro Zoll angibt.

Das Feld *dmColor* wählt zwischen Farb- und Monochromdruck, wobei die *dmcolor_*-Konstanten eingesetzt werden. *dmDuplex* wählt mittels der *dmdup_*-Konstanten ein- oder zweiseitigen Druck.

Das Feld *dmDriverData* enthält treiberspezifische Daten, die vom Treiber definiert wurden.

TDrawItemStruct-Typ **WinTypes**

Deklaration

```
TDrawItemStruct = record
  CtlType: Word;
  CtlID: Word;
  itemID: Word;
  itemAction: Word;
  itemState: Word;
  hwndItem: HWnd;
  hDC: HDC;
  rcItem: TRect;
  itemData: LongInt;
end;
```

Beschreibung

TDrawItemStruct-Records enthalten Daten zum Zeichnen benutzerdefinierter Steuerelemente. Der Eigner des Steuerelements erhält einen Zeiger auf einen *TDrawItemStruct*-Record im Parameter *lParam* der Botschaft *wm_DrawItem*.

Das Feld *CtlType* ist der Steuerelementtyp, der von einer der *odt_*-Konstanten festgelegt wird.

CtlID ist die Steuerelement-ID (nicht für Menüs verwendet) und *itemID* die Menüoptions-ID, abhängig vom Steuerelement. Diese kann für leere Listen oder Kombinationsfenster -1 sein.

Das Feld *itemAction* definiert mit den *oda_*-Konstanten Zeichenaktionen, um anzugeben, wann und wie das Steuerelement gezeichnet werden soll.

Das Feld *itemState* beschreibt mit den *ods_*Konstanten den Status des Eintrags, nachdem er gezeichnet wurde.

Das Feld *hwndItem* ist das Fenster-Handle des Steuerelements bzw. für ein Menü das Handle des Menüs, das den Eintrag enthält. *hDC* ist das Handle des Gerätekontextes, der benutzt werden muß, wenn dieses Steuerelement gezeichnet wird.

Das Feld *rcItem* ist das umgrenzende Rechteck (Record *TRect*) des Steuerelements im Gerätekontext. Windows schneidet benutzerdefinierte Steuerelemente an dieser Grenze ab, benutzerdefinierte Menüoptionen können diese Grenze jedoch überschreiten.

Das Feld *itemData* kann zwei verschiedene Werte enthalten. Entweder den Wert eines Eintrags in einem benutzerdefinierten Listen- oder Kombinationsfensters, der von einer der Botschaften *cb_AddString, cb_InsertString, lb_AddString* oder *lb_InsertString* geliefert wurde, oder den LongInt-Wert, der im Parameter *NewItem* des Aufrufs von *InsertMenu* an die Menüoption übergeben wurde. *itemData* ist für benutzerdefinierte Schalter undefiniert.

TFarProcTyp — WinTypes

Deklaration
```
TFarProc = Pointer;
```

Beschreibung *TFarProc* ist normalerweise ein Zeiger auf eine Prozedur.

TGlobalHandle-Typ — WinTypes

Deklaration
```
TGlobalHandle ╪Handle;
```

Beschreibung *TGlobalHandle* ist identisch mit *THandle* und kann verwendet werden, um im Quelltext kenntlich zu machen, daß es sich um ein Handle zu einem globalen Element handelt.

THandle-Typ — WinTypes

Deklaration
```
THandle = Word;
```

Beschreibung *THandle* definiert einen generischen Handle-Typ.

THandleTable-Typ — WinTypes

Deklaration
```
THandleTable = record
  objectHandle:    array [0..0]  of THandle;
end;
```

Beschreibung *THandleTable* ist ein Array von Handles, das dazu verwendet wird, eine Anzahl von Zeichenwerkzeugen zu speichern.

TLocalHandle-Typ — WinTypes

Deklaration
```
TLocalHandle = THandle;
```

Beschreibung *TLocalHandle* ist identisch mit *THandle* und kann dazu verwendet werden, im Quelltext kenntlich zu machen, daß es sich um ein Handle zu einem lokalen Element handelt.

TLogBrush-Typ — WinTypes

Deklaration
```
TLogBrush = record
  lbStyle: Word;
  lbColor: LongInt;
  lbHatch: Integer;
end;
```

Beschreibung *TLogBrush*-Records werden verwendet, um die Information zur Erzeugung eines virtuellen Pinsels mit der Funktion *CreateBrushIndirect* zu speichern.

Das Feld *lbStyle* enthält eine der *bs_*-Pinselstilkonstanten, die anzeigen, ob der Pinsel kompakt, leer oder schraffiert ist bzw. ob er ein Muster hat.

Das Feld *lbColor* ist ein *TColorRef*-Record. Die Farbe wird ignoriert, wenn der Stil des Pinsels leer oder gemustert ist. Wenn der Stil *bs_DIBPattern* ist, muß das niederwertige Word eine der *DIB_*-Konstanten enthalten. Diese zeigen an, ob die bezeichneten Farben explizit vorliegen oder in die aktuelle Farbpalette indiziert sind.

Das Feld *lbHatch* gibt einen Schraffurstil an. Abhängig vom Stil des Pinsels kann *lbHatch* folgendes enthalten:

Stil	*lbHatch* enthält
bs_DIBPattern	Das Handle eines gepackten geräteunabhängigen Bitmaps.
bs_Hatched	Eine der *hs_*-Schraffurstilkonstanten, die die Ausrichtung der Schraffur anzeigen.
bs_Hollow	Ignoriert.
bs_Pattern	Das Handle des Bitmaps, das das Muster definiert.
bs_Solid	Ignoriert.

TLogFont-Typ **WinTypes**

Deklaration

```
TLogFont = record
  lfHeight: Integer;
  lfWidth: Integer;
  lfEscapement: Integer;
  lfOrientation: Integer;
  lfWeight: Integer;
  lfItalic: Byte;
  lfUnderline: Byte;
  lfStrikeOut: Byte;
  lfCharSet: Byte;
  lfOutPrecision: Byte;
  lfClipPrecision: Byte;
  lfQuality: Byte;
  lfPitchAndFamily: Byte;
  lfFaceName:    array [0..lf_FaceSize - 1]    of  Byte;
end;
```

Beschreibung *TLogFont*-Records enthalten die Attribute einer virtuellen Schrift zur Verwendung durch die Funktion *CreateFontIndirect*.

lfHeight und *lfWidth* geben die durchschnittliche Höhe und Breite der Schrift an. *lfEscapement* und *lfOrientation* sind der Schrittschaltungswinkel und der Neigungswinkel des Textes von der x-Achse aus gemessen. Beide werden in Zehntelgraden und entgegen dem Uhrzeigersinn angegeben.

Das Feld *lfWeight* gibt das Gewicht der Schrift in einge-

schwärzten Punkten pro Tausend an. Der Wert kann zwischen 0 und 1000 liegen. Zum Beispiel gibt 400 einen normalen Grauwert an, 700 bezeichnet eine Fettschrift. Die tatsächlichen Werte variieren in Abhängigkeit vom Schriftbild. Ein Wert von 0 zeigt an, daß ein Standardgewicht benutzt werden soll.

Die Felder *lfItalic, lfUnderline* und *lfStrikeOut* sind normalerweise 0. Wenn ihr Wert ungleich 0 ist, wird dadurch kursive, unterstrichene oder durchgestrichene Schrift angegeben.

lfCharSet ist einer der drei vordefinierten Zeichensätze *ANSI_CharSet, OEM_CharSet* oder *Symbol_CharSet*. Es können auch andere Zeichensätze definiert werden.

Das Feld *lfOutPrecision* enthält eines der *out_*-Schriftgenauigkeits-Flags, voreingestellt ist der Wert *out_Default_Precis*. *lfClipPrecision* gibt die Clipping-Genauigkeit der Schrift an, die von den Schriftgenauigkeits-Flags definiert wird, voreingestellt ist der Wert *clip_Default_Precis*.

Das Feld *lfQuality* enthält eines der Schriftqualitäts-Flags *Default_Quality, Draft_Quality* oder *Proof_Quality*. *lfPitchAndFamily* ist die Kombination eines Zeichenabstand-Flags (entweder *Default_Pitch, Fixed_Pitch* oder *Variable_Pitch*) und eines Schriftfamilien-Flags (*ff_Roman* oder *ff_Script*). *lfFaceName* enthält die Bezeichnung der Schrift in einem nullterminierten String. Ist *lfFaceName* **nil**, so verwendet die GDI einen Standardschrifttyp.

TLogPalette-Typ

WinTypes

Deklaration

```
TLogPalette = record
  palVersion: Word;
  palNumEntries: Word;
  palPalEntry:    array [0..0]  of TPaletteEntry;
end;
```

Beschreibung

TLogPalette-Records enthalten Daten zur Definition einer virtuellen Palette, wie sie von der Funktion *CreatePalette* verwendet werden.

Das Feld *palVersion* gibt die Version von Windows für die Struktur an, gegenwärtig ist dies $0300. *palNumEntries* ist die Anzahl von Einträgen in der Palette, *palPalEntry* ein Array von *TPaletteEntry*-Records – ein Element für jeden Eintrag in der Palette.

TLogPen-Typ

WinTypes

Deklaration

```
TLogPen = record
  lopnStyle: Word;
  lopnWidth: TPoint;
```

```
  lopnColor: LongInt;
end;
```

Beschreibung *TLogPen*-Records enthalten die Attribute eines virtuellen Stifts und werden von der Funktion *CreatePenIndirect* verwendet.

Das Feld *lopnStyle* bezeichnet den Stiftstil, wozu eine der *ps_*-Konstanten verwendet wird. *lopnWidth* ist die Breite des Stifts in virtuellen Einheiten, *lopnColor* die Stiftfarbe.

TMDICreateStruct-Typ

WinTypes

Deklaration

```
TMDICreateStruct = record
  szClass: PChar;
  szTitle: PChar;
  hOwner: THandle;
  x, y: Integer;
  cx, cy: Integer;
  style: LongInt;
  lParam: LongInt;
end;
```

Beschreibung *TMDICreateStruct*-Records enthalten Daten für die Erzeugung eines untergeordneten MDI-Fensters. Der Parameter *lParam* einer Botschaft *wm_Create* enthält einen *TCreateStruct*-Record, dessen Feld *lpCreateParams* auf einen *TMDICreateStruct*-Record zeigt, der von einer Botschaft *wm_MDICreate* geliefert wurde.

Das Feld *szClass* zeigt auf die Fensterklasse des untergeordneten Fensters, *szTitle* auf den Fenstertitel. *hOwner* ist das Instanz-Handle der Anwendung, die das Fenster erzeugt.

x und *y* sind die anfänglichen x- und y-Koordinaten des untergeordneten Fensters, *cx* und *cy* die anfängliche Breite und Höhe. Sind alle diese Werte auf *cw_UseDefault* gesetzt, so wird ein jeweils passender Standardwert eingesetzt.

Das Feld *style* enthält zusätzliche Stile für das Fenster, diese können *ws_Minimize, ws_Maximize, ws_HScroll* oder *ws_VScroll* sein.

lParam wird von der Anwendung definiert.

TMeasureItemStruct-Typ

WinTypes

Deklaration

```
TMeasureItemStruct = record
  CtlType: Word;
  CtlID: Word;
  itemID: Word;
  itemWidth: Word;
  itemHeight: Word;
  itemData: LongInt;
end;
```

Beschreibung *TMeasureItemStruct*-Records enthalten die Abmessungen eines benutzerdefinierten Steuerelements.

Eine *wm_MeasureItem*-Botschaft enthält in seinem Parameter *lParam* einen Zeiger auf einen *TMeasureItemStruct*-Record. Wenn dessen Felder nicht richtig belegt werden, arbeiten die besitzergezeichneten Steuerelemente nicht korrekt.

Das Feld *CTLType* ist der Steuerelementtyp, der durch eine der *odt_*-Konstanten festgelegt wird. *CtlID* ist die Steuerelement-ID-Zahl (dies gilt nicht für Menüs). *itemID* ist die Menüoptions-ID für Menüs.

Die Felder *itemWidth* und *itemHeight* enthalten die Breite bzw. die Höhe der Option.

Das Feld *itemData* kann zwei Werte enthalten: Entweder einen Eintrag in einem besitzergezeichneten Listen- oder Kombinationsfenster, der von einer der Botschaften *cb_AddString*, *cb_InsertString*, *lb_AddString* oder *lb_InsertString* geliefert wurde, oder den LongInt-Wert, der an die Menüoption über den Parameter *NewItem* des Aufrufs von *InsertMenu*, übergeben wurde.

itemData ist für besitzergezeichnete Schalter undefiniert.

TMenuItemTemplateHeader-Typ

WinTypes

Deklaration

```
TMenuItemTemplateHeader = record
  versionNumber: Word;
  offset: Word;
end;
```

TMetaFilePict-Typ

WinTypes

Deklaration

```
TMetaFilePict = record
  mm: Integer;
  xExt: Integer;
  yExt: Integer;
  hMF: THandle;
end;
```

Beschreibung *TMetaFilePict*-Records definieren das Format der Metafile-Bilder, die verwendet werden, um Metafile-Daten über die Zwischenablage auszutauschen.

Das Feld *mm* enthält den Abbildungsmodus, in dem das Bild gezeichnet wurde. Wenn der Abbildungsmodus nicht *mm_Isotropic* oder *mm_Anisotropic* ist, bezeichnen *xExt* und *yExt* Breite und Höhe des Rechtecks, in dem das Bild gezeichnet wurde. Andernfalls enthalten die Felder vorgeschlagene Größen (*mm_Anisotropic*) oder relative Größen (*mm_Isotropic*). *hMF* ist ein Handle eines Speicher-Metafiles.

TMetaHeader-Typ **WinTypes**

Deklaration
```
TMetaHeader = record
  mtType: Word;
  mtHeaderSize: Word;
  mtVersion: Word;
  mtSize: LongInt;
  mtNoObjects: Word;
  mtMaxRecord: LongInt;
  mtNoParameters: Word;
end;
```

Beschreibung *TMetaHeader*-Records definieren das Format des Headers eines Metafiles. Ein Metafile besteht aus einem *TMetaHeader*-Record, gefolgt von einer Liste von Metafile-Records, die normalerweise vom Typ *TMetaRecord* sind.

mtType kann zwei Werte enthalten: 1 zeigt an, daß sich das Metafile im Arbeitsspeicher befindet, 2 zeigt an, daß es auf einer Platte abgelegt ist. Das Feld *mtHeaderSize* gibt die Größe des Headers in Byte an. *mtVersion* ist die Windows-Versionsnummer, diese ist derzeit $0300. *mtSize* ist die Größe der Datei in Byte. *mtNoObjects* kennzeichnet die maximale Anzahl von Objekten, die das Metafile enthalten kann, und *mtMaxRecord* die Größe (in Word) des längsten Metafile-Records.

mtNoParameters wird gegenwärtig nicht verwendet.

TMetaRecord-Typ **WinTypes**

Deklaration
```
TMetaRecord = record
  rdSize: LongInt;
  rdFunction: Word;
  rdParam:   array [0..0]  of Word;
end;
```

Beschreibung *TMetaRecord*-Records definieren typische Metafile-Records. Eine Liste solcher Records folgt auf den Header eines Metafiles.

Das Feld *rdSize* gibt die Größe des Records in Worten an.

rdFunction ist die Funktionsnummer, die durch eine *meta_*-Konstante angegeben wird.

rdParm ist ein Array von Parametern des Typs *Word* für die Funktion, die in umgekehrter Reihenfolge gespeichert sind, in der sie an die Funktion übergeben werden.

TMsg-Typ **WinTypes**

Deklaration
```
TMsg = record
  hwnd: HWnd;
  message: Word;
```

```
  wParam: Word;
  lParam: LongInt;
  time: LongInt;
  pt: TPoint;
end;
```

Beschreibung *TMsg*-Records enthalten Botschaftsdaten, die an die Anwendungen durch Windows verteilt werden. Die Information wird an die betreffenden Elemente (Fenster) zur Verarbeitung weitergeleitet. Die verschiedenen Felder enthalten wichtige Informationen für die Anwendung.

Hinweis: Alle Windows-Botschaften mit ihren jeweiligen Parametern sind in Kapitel 3 »Windows-Botschaften« aufgelistet.

Das Feld *hwnd* ist das Handle des Fensters, das die Botschaft empfängt.

message ist die Nummer der Botschaft.

Die Felder *wParam* und *lParam* enthalten Information für das Fenster in Word- und LongInt-Länge. Diese Information variiert in Abhängigkeit vom Wert von *message*. Die Werte der Parameter jeder Botschaft sind in Kapitel 3 »Windows-Botschaften« beschrieben.

Das Feld *time* enthält die Zeit, zu der die Botschaft übermittelt wurde, *pt* die Bildschirmposition des Mauszeigers zu diesem Zeitpunkt.

TMultiKeyHelp-Typ

WinTypes

Deklaration

```
TMultiKeyHelp = record
  mkSize:Word;
  mkKeyList: Byte;
  szKeyPhrase:    array [0..0]  of Byte;
end;
```

Beschreibung *TMultiKeyHelp*-Records enthalten für das Windows-Hilfesystem einen Index auf eine Tabelle von Schlüsselworten und einen Ausdruck, nach dem gesucht werden soll.

Das Feld *mkSize* enthält die Länge des Records. *mkKeyList* ist ein Zeichen, das anzeigt, welche Schüsselworttabelle durchsucht werden soll, *szKeyPhrase* ein nullterminierter String, der das aufzufindende Schlüsselwort enthält.

TOFStruct-Typ

WinTypes

Deklaration

```
TOFStruct = record
  cBytes: Byte;
  fFixedDisk: Byte;
  nErrCode: Word;
```

```
  reserved:   array [0..3] of Byte;
  szPathName:   array [0..127] of Char;
end;
```

Beschreibung *TOFStruct*-Records enthalten Information über eine Datei, die aufgerufen wurde, als die Datei geöffnet wurde.

Das Feld *cBytes* enthält die Länge des Records. *fFixedDisk* ist ungleich 0, wenn sich die Datei auf einer Festplatte befindet, andernfalls ist es 0. *nErrCode* enthält den DOS-Fehlercode, wenn die Funktion *OpenFile* fehlgeschlagen ist (*OpenFile* liefert dann -1 zurück). *reserved* enthält vier Bytes, die für eine zukünftige Verwendung durch Windows reserviert sind.

Das Feld *szPathName* ist ein nullterminierter String, der den vollständigen Pfadnamen der Datei enthält.

TPaintStruct-Typ

WinTypes

Deklaration

```
TPaintStruct = record
  hdc: HDC;
  fErase: Bool;
  rcPaint: TRect;
  fRestore: Bool;
  fIncUpdate: Bool;
  rgbReserved:   array [0..15] of Byte;
end;
```

Beschreibung *TPaintStruct*-Records enthalten Information, die von einer Anwendung zum Zeichnen der Client-Bereiche ihrer Fenster benutzt werden.

Ein Großteil dieser Information ist für die interne Verwendung durch Windows reserviert, einige Felder können jedoch vom Benutzer belegt werden.

Das Feld *hdc* ist das Handle des Bildschirmkontexts, in dem die Zeichnung erscheinen soll. *fErase* zeigt an, ob der Hintergrund neugezeichnet wurde, der Wert 0 bedeutet, daß er nicht neugezeichnet wurde. *rcPaint* definiert das Rechteck, in dem die Zeichnung Platz finden soll.

Alle anderen Felder sind für dem internen Gebrauch durch Windows reserviert.

TPaletteEntry-Typ

WinTypes

Deklaration

```
TPaletteEntry: record
  peRed: Byte;
  peGreen: Byte;
  peBlue: Byte;
  peFlags: Byte;
end;
```

Beschreibung Der Recordtyp *TPaletteEntry* definiert einen Eintrag in eine virtuelle Palette, so wie von *TLogPalette* definiert.

Die Felder *peRed*, *peGreen* und *peBlue* repräsentieren jeweils die Intensitäten von Rot, Grün und Blau im Paletteneintrag. *peFlags* enthält Information darüber, wie die Palette eingesetzt werden soll. Es kann 0 oder eines der *pc_*-Flags *pc_Explicit, pc_NoCollapse* oder *pc_Reserved* sein.

TPattern-Typ

WinTypes

Deklaration
```
TPattern = TLogBrush;
```

Beschreibung *TPattern* ist eine andere Bezeichnung für *TLogBrush*. Wenn ein virtueller Pinsel für gemustertes Flächenfüllen benutzt wird, können Sie die Bezeichnung *TPattern* wählen, um dies zu kennzeichnen.

TPoint-Typ

WinTypes

Deklaration
```
TPoint = record
  x: Integer;
  y: Integer;
end;
```

Beschreibung Der Record *TPoint* ist zwar sehr einfach, aber auch sehr nützlich. Er bezeichnet die x- und y-Koordinaten (jeweils in den Feldern *x* und *y*) eines Punktes auf dem Bildschirm oder in einem Fenster.

TRect-Typ

WinTypes

Deklaration
```
TRect = record
  left: Integer;
  top: Integer;
  right: Integer;
  bottom: Integer;
end;
```

Beschreibung Der Record *TRect* definiert einen rechteckigen Bereich durch die Kennzeichnung der oberen linken und der unteren rechten Ecke.

Die Felder *left* und *top* sind die x- und y-Koordinaten der linken oberen Ecke, *right* und *bottom* die x- und y-Koordinaten der unteren rechten Ecke.

Das Rechteck darf in Breite oder Höhe 32768 Einheiten nicht überschreiten.

TRGBQuad-Typ **WinTypes**

Deklaration
```
TRGBQuad = record
  rgbBlue: Byte;
  rgbGreen: Byte;
  rgbRed: Byte;
  rgbReserved: Byte
end;
```

Beschreibung *TRGBQuad*-Records enthalten RGB-Farbdaten zur Verwendung durch Bitmaps wie im Feld *bmiColors* eines Records *TBitmapInfo*.

Die Felder *rgbBlue, rgbGreen* und *rgbRed* repräsentieren jeweils die Intensitäten von Blau, Grün und Rot im Bitmap-Pixeleintrag. *rgbReserved* wird nicht verwendet und muß 0 sein.

TRGBTriple-Typ **WinTypes**

Deklaration
```
TRGBTriple = record
  rgbtBlue: Byte;
  rgbtGreen: Byte;
  rgbtRed: Byte;
end;
```

Beschreibung Der Record *TRGBTriple* bezeichnet die RGB-Farbintensitäten für Bitmaps, so wie das Feld *bmciColors* eines Records *TBitmapCoreInfo*.

Die Felder *rgbtBlue, rgbtGreen* und *rgbtRed* repräsentieren jeweils die Intensitäten von Blau, Grün und Rot im Bitmap-Pixeleintrag.

TTextMetric-Typ **WinTypes**

Deklaration
```
TTextMetric = record
  tmHeight: Integer;
  tmAscent: Integer;
  tmDescent: Integer;
  tmInternalLeading: Integer;
  tmExternalLeading: Integer;
  tmAveCharWidth: Integer;
  tmMaxCharWidth: Integer;
  tmWeight: Integer;
  tmItalic: Byte;
  tmUnderlined: Byte;
  tmStruckOut: Byte;
  tmFirstChar: Byte;
  tmLastChar: Byte;
  tmDefaultChar: Byte;
```

```
  tmBreakChar: Byte;
  tmPitchAndFamily: Byte;
  tmCharSet: Byte;
  tmOverhang: Integer;
  tmDigitizedAspectX: Integer;
  tmDigitizedAspectY: Integer;
end;
```

Beschreibung Der Record *TTextMetric* enthält zahlreiche Felder, die eine physikalische Schrift in Einheiten beschreiben, die vom Abbildungsmodus des Bildschirmkontexts abhängen. Die Records *TTextMetrics* werden von den Funktionen *GetDeviceCaps* und *GetTextMetrics* verwendet.

Das Feld *tmHeight* ist die Zeichenhöhe der Schrift, die sich aus der Summe der Oberlänge (*tmAscent*) und der Unterlänge (*tmDescent*) unterhalb der Grundlinie ergibt.

Die Felder *tmInternalLeading* und *tmExternalLeading* geben das Ausmaß des zulässigen Leerraums außerhalb des Bereichs von *tmHeight* an. *tmInternalLeading* ist der Leerraum innerhalb der Grenzen, *tmExternalLeading* wird zwischen den Textzeilen hinzuaddiert. Beide Felder können auf 0 gesetzt werden.

Die Felder *tmAveCharWidth* und *tmMaxCharWidth* geben die durchschnittliche und die maximale Zeichenbreite für die Zeichen in der Schrift an.

tmWeight ist das Gewicht der Schrift. *tmItalic, tmUnderlined* und *tmStruckOut* kennzeichnen jeweils kursiven, unterstrichenen und durchgestrichenen Text.

Der Bereich definierter Zeichen in der Schrift wird von *tmFirstChar* und *tmLastChar* angegeben, wobei Zeichen außerhalb dieses Bereiches durch das in *tmDefaultChar* angegebene Zeichen ersetzt werden. *tmBreakChar* ist das Zeichen, das die Wortzwischenräume für Ausrichtungszwecke abgrenzt.

Der Zeichenabstand, die Familie und der Zeichensatz der Schrift werden von *tmPitchAndFamily* und *tmCharSet* definiert. Das niederwertigste Bit in *tmPitchAndFamily* bestimmt den Zeichenabstand der Schrift: fest, wenn das Bit gelöscht ist, variabel, wenn das Bit gesetzt ist. Die vier höherwertigen Bits legen die Schriftenfamilie fest, die mit den *ff_*-Schriftenfamilien-Flags gesetzt oder abgefragt werden kann. Der Zeichensatz ist in *tmCharSet* enthalten, hier werden die Schriftzeichensatzflags eingesetzt.

Das Feld *tmOverhang* enthält die zusätzliche Breite, die bei manchen synthetisierten Schriften pro String hinzugefügt wird, beispielweise wenn aus einer normalen eine fette Schrift gemacht wird.

Das horizontale und vertikale Seitenverhältnis des Gerätes, für das die Schrift entworfen wurde, ist in den Feldern *tmDigitizedAspectX* bzw. *tmDigitizedAspectY* gespeichert.

TWndClass-Typ **WinTypes**

Deklaration

```
TWndClass = record
  style: Word
  lpfnWndProc: TFarProc;
  cbClsExtra: Integer;
  cbWndExtra: Integer;
  hInstance: THandle;
  hIcon: HIcon;
  hCursor: HCursor;
  hbrBackground: HBrush;
  lpszMenuName: PChar;
  lpszClassName: PChar;
end;
```

Beschreibung

TWndClass-Records enthalten die Attribute einer Fensterklasse – auch bezeichnet als Registrierungsattribute – so wie sie von der Funktion *RegisterClass* registriert werden.

Das Feld *Style* enthält den Klassenstil. Es kann eine oder eine Kombination der *cs_*-Klassenstilkonstanten beinhalten.

Das Feld *lpfnWndProc* zeigt auf die Fensterfunktion, die Routine, die Botschaften empfängt und verarbeitet.

cbClsExtra ist die Anzahl von Bytes, die am Ende des Records *TWndClass* angeordnet werden. Diese werden als die Extrabytes der Klasse bezeichnet. Auf diese kann mit den Funktionen *GetWindowLong* oder *GetWindowWord* zugegriffen werden, oder sie können mit den Funktionen *SetWindowLong* oder *SetWindowWord* gesetzt werden.

cbWndExtra gibt die Anzahl der Bytes an, die am Ende der Fensterklasse angeordnet werden sollen.

hInstance ist ein Instanz-Handle, das das Klassenmodul kennzeichnen muß. Es darf nicht 0 sein.

Die Felder *hIcon, hCursor* und *hbrBackground* sind jeweils die Handles des Symbols des Mauszeigers und der Hintergrundfarbe der Klasse. Die Hintergrundfarbe sollte ein Farbwert (eine der Standardsystemfarben, bezeichnet durch eine der um 1 inkrementierten *color_*-Konstanten) oder das Handle eines Pinsels zum Hintergrundzeichnen sein. Wenn *hbrBackground* 0 ist, muß der Hintergrund der Anwendung immer dann gezeichnet werden, wenn ihr Client-Bereich gezeichnet wird. Ob dies notwendig ist, kann durch Bearbeiten der Botschaft *wm_EraseBkgnd* oder durch Abfragen des Feldes *fErase* des von *BeginPaint* erzeugten *TPaintStruct*-Records festgestellt werden.

Die Felder *lpszMenuName* und *lpszClassName* zeigen beide auf nullterminierte Strings, die die Ressourcenbezeichnung des Klassenmenüs bzw. die Klassenbezeichnung sind.

II. ObjectWindows-Referenz

5. Verzeichnis der Objekte

Dieses Kapitel enthält eine alphabetische Aufzählung aller Standardobjekttypen von ObjectWindows mit den Erklärungen ihrer Aufgabe und ihrer Anwendung, ihren Feldern, Methoden und Farbpaletten.

Um Informationen über ein bestimmtes Objekt zu finden, sollte man bedenken, daß viele der Eigenschaften eines Objekts in der Hierarchie von Vorfahrensobjekten ererbt sind. Um diese Information nicht endlos zu vervielfachen, dokumentiert dieses Kapitel nur Felder und Methoden, die für ein bestimmtes Objekt *neu* oder *verändert* sind.

Wenn Sie z.B. etwas über das Feld *Parent* eines Objekts *TEdit* wissen möchten, müssen Sie zuerst bei den Feldern von *TEdit* nachsehen, wo aber *Parent* nicht aufgeführt ist. Danach suchen Sie *TStatic*, den direkten Vorfahren von *TEdit* in der Hierarchie – *Parent* ist hier ebenfalls nicht aufgelistet. Also überprüfen Sie als nächstes *TControl*, den direkten Vorfahren von *TStatic*. Dieses Verfahren setzen Sie solange fort, bis Sie schließlich zu *TWindowsObject* gelangen. Hier finden Sie die vollständige Information über *Parent*, das unverändert von *TEdit* ererbt ist. Die Beschreibung jedes Objekts ist in folgendem Format aufgebaut:

TSample-Objekt — Unit des Objekts

Beschreibung	Zu Beginn steht ein allgemeiner Überblick über das Objekt, seine Verwandtschaft mit anderen Objekten und die allgemeine Verwendung
Felder	Dieser Abschnitt beschreibt alle Felder für jedes Objekt in alphabetischer Reihenfolge.
AField	`AField: SomeType;` *AField* ist ein Feld, das einige Informationen über das Objekt Sample enthält. Es wird erklärt, wie es arbeitet, was es bedeutet und wie man es anwendet. Querverweis: Verwandte Felder, Methoden, Objekte, globale Funktionen, usw.
AnotherField	`AnotherField: Word;` *AnotherField* enthält ähnliche Informationen wie *AField*.

Methoden Dieser Abschnitt listet alle Methoden auf, die für das Objekt entweder neu definiert sind oder die ererbte Methoden überschreiben.

Init

```
constructor Init(AParameter: SomeType);
```

Init erzeugt ein neues Objekt Sample und setzt das Feld *AField* auf *AParameter*.

Zilch

```
procedure Zilch; virtual;
```

Die Methode *Zilch* veranlaßt das Objekt Sample dazu, eine Aktion auszuführen.

TApplication **WObjects**

Beschreibung *TApplication* stellt die Struktur für eine ObjectWindows-Anwendung zur Verfügung. Alle ObjectWindows-Anwendungen leiten einen Objekttyp von *TApplication* ab, vor allem um ein Hauptfenster eines benutzerdefinierten Objekttyps zu konstruieren.

Felder

HAccTable

```
HAccTable: THandle;
```

HAccTable enthält ein Handle zu einer Ressource einer Accelerator-Tabelle, die für die Anwendung definiert ist.

KBHandler-Wnd

```
KBHandlerWnd: PWindowsObject;
```

KBHandlerWnd zeigt auf das aktive Fenster, wenn der Mechanismus zur Tastaturverwaltung des Fensters eingeschaltet ist. Dieser Mechanismus ermöglicht es einem Fenster, mit Steuerelementen Tastatureingaben so wie ein Dialogfenster zu verarbeiten. *KBHandlerWnd* ist **nil**, wenn der Mechanismus für das aktive Fenster ausgeschaltet ist.

MainWindow

```
MainWindow: PWindowsObject;
```

MainWindow zeigt auf das überlappte Hauptfenster der Anwendung, das von der Methode *InitMainWindow* Ihres Anwendungstyps instantiiert werden sollte.

Name

```
Name: PChar;
```

Name enthält den Namen der Anwendung. Dieser kann intern vom Code der Anwendung benutzt werden.

Status

```
Status: Integer;
```

Status kennzeichnet den aktuellen Status der laufenden Anwendung. Diese läuft fehlerfrei, wenn *Status* größer oder gleich 0 ist. Zu den Fehlerwerten gehört *em_InvalidMainWindow*, das ein ungültiges Hauptfensterobjekt anzeigt.

Methoden

Init

```
constructor Init(AName: PChar);
```

Initialisiert das Anwendungsobjekt. Ruft zuerst *TObject.Init* auf und setzt anschließend die globale Variable *Application* auf *@Self*, das Feld *Name* auf *AName* und die Felder *HAccTable* und *Status* auf 0. Die Felder *MainWindow* und *KBHandlerWnd* werden auf **nil** initialisiert.

Ist dies die erste Instanz dieser Anwendung, ruft *Init InitApplication* auf. Ist *InitApplication* erfolgreich (d.h., das Feld *Status* ist 0), wird *InitInstance* aufgerufen.

Sie können diese Methode überschreiben, um zum Beispiel eine Accelerator-Tabelle für Ihre Anwendung zu laden. Stellen Sie sicher, daß Sie diese Methode von jeder Methode aus aufrufen, die sie überschreibt.

Querverweis: *TApplication.InitApplication, TApplication.InitInstance, TObject.Init*

Done

```
destructor Done; virtual;
```

Gibt die zur Anwendung gehörigen Objekte durch Freigabe von *MainWindow* frei und ruft dann *TObject.Done* auf, um die Anwendung zu beenden.

Querverweis: *TObject.Done*

CanClose

```
function CanClose: Boolean; virtual;
```

Diese Methode liefert *True* zurück, wenn es für die Anwendung keinen Grund gibt, nicht zu schließen. Standardmäßig ruft sie *CanClose* ihres Hauptfensters auf und gibt deren Rückgabewert zurück. Diese Methode wird selten überschrieben werden; die Vorgehensweise beim Schließen kann in der Methode *CanClose* des Hauptfensters überschrieben werden.

Querverweis: *TWindowsObject.CanClose, TWindowsObject.WMDestroy*

Error

```
procedure Error(ErrorCode: Integer); virtual;
```

Error verarbeitet Fehler, die von dem in *ErrorCode* übergebenen Fehlerwert bezeichnet werden. Diese Fehler können vom Anwendungsobjekt oder jedem Fenster oder Dialogobjekt generiert werden. *ErrorCode* kann ein von Ihnen definierter oder einer der folgenden Fehlerwerte sein, die von ObjectWindows erkannt und gemeldet wurden:

em_InvalidWindow
em_OutOfMemory
em_InvalidClient
em_InvalidChild
em_InvalidMainWindow

Die *em_*-Konstanten werden in Kapitel 6 »Außerhalb der Objekte« beschrieben.

Error stellt den Fehlercode in einem Meldungsfenster dar und fragt den Benutzer, ob weitergemacht werden soll. Wenn nicht, stoppt die Programmausführung.

ExecDialog

```
function ExecDialog(ADialog: PWindowsObject): Integer;
virtual;
```

Diese Methode führt nach einer Überprüfung von *ValidWindow* das modale Dialogobjekt aus, das in *ADialog* übergeben wurde, indem sie die Methode *Execute* des Dialogobjekts aufruft. Wenn der Speicherplatz nicht ausreicht oder der Dialog nicht ausgeführt werden kann, gibt *ExecDialog* das Objekt frei und liefert einen negativen Fehlerstatus zurück.

Querverweis: *TDialog.Execute*

InitApplication

```
procedure InitApplication; virtual;
```

Führt die notwendige Initialisierung für die erste Instanz der Anwendung durch. *TApplication.InitApplication* macht standardmäßig nichts . Ihr Anwendungsobjekttyp kann *InitApplication* überschreiben, um anwendungsspezifische Initialisierungen durchzuführen.

InitInstance

```
procedure InitInstance; virtual;
```

Führt die notwendige Initialisierung für jede Instanz der Anwendung durch. *TApplication.InitInstance* ruft *InitMainWindow* auf, erzeugt das Hauptfenster durch den Aufruf von *MakeWindow* und stellt es mit *Show* dar. Im Fehlerfall wird das Feld *Status* auf *em_InvalidMainWindow* gesetzt. Wenn Sie diese Methode überschreiben, müssen Sie *TApplication.InitInstance* explizit aufrufen.

Querverweis: *TApplication.InitMainWindow*

InitMain Window

```
procedure InitMainWindow; virtual;
```

Per Voreinstellung instantiiert *TApplication.InitMainWindow* ein generisches *TWindow*-Objekt ohne Titel. Überschreiben Sie *InitMainWindow*, um ein nützliches Hauptfensterobjekt zu konstruieren, und speichern Sie es in *MainWindow*. Ein typisches Beispiel:

```
procedure  MyApplication.InitMainWindow;
begin
  MainWindow := New(PMyWindow, Init('Window Caption'));
end ;
```

MakeWindow

```
function MakeWindow (AWindowsObject: PWindowsObject):
PWindowsObject; virtual;
```

Versucht nach einer Überprüfung des Sicherheitsbereichs ein Fenster oder ein nichtmodales Dialogelement zu erzeugen, das dem in *AWindowsObject* übergebenen Objekt zugeordnet ist. Wenn der Speicher nicht ausreicht (*LowMemory* gibt *True* zurück), oder wenn das Fenster oder der Dialog nicht erzeugt werden kann, gibt *MakeWindow* das Objekt frei und liefert **nil** zurück. Andernfalls gibt die Funktion *AWindowsObject* zurück.

Querverweis: *TWindow.Create, LowMemory*

MessageLoop `procedure MessageLoop; virtual;`

Arbeitet die Botschaftsschleife der Anwendung ab, die während der gesamten Dauer der Anwendung läuft. *TApplication.MessageLoop* ruft *ProcessAppMsg* auf, um spezielle Botschaften für nichtmodale Dialoge, normale und MDI-Accelerator-Tasten zu bearbeiten. Spezielle Botschaftsverarbeitung sollte besser in *ProcessAppMsg* anstatt in *MessageLoop* vorgenommen werden.

Querverweis: *TApplication.ProcessAppMsg*

ProcessAppMsg `function ProcessAppMsg(var Message: TMsg): Boolean; virtual;`

Prüft auf spezielle Verarbeitung für nichtmodale Dialog-, Accelerator- und MDI-Accelerator-Botschaften. Ruft *ProcessDldMsg, ProcessMDIAccels* und *ProcessAccels* auf und gibt *True* zurück, wenn eine dieser speziellen Botschaften eintrifft.

Wenn Ihre Anwendung keine nichtmodalen Dialoge erzeugt, nicht auf Accelerator-Tasten reagiert und keine MDI-Anwendung ist, können Sie die Leistungsfähigkeit erhöhen, indem Sie diese Methode überschreiben und einfach *False* zurückgeben.

ProcessDlgMsg `function ProcessDlgMsg(var Message: TMsg): Boolean; virtual;`

Übernimmt die Bearbeitung nichtmodaler Dialog- und Fensterbotschaften zur Verwaltung von Tastatureingaben für Steuerelemente. Wenn Ihre Anwendung keine nichtmodalen Dialoge oder Fenster mit Steuerelementen erzeugt, können Sie die Leistungsfähigkeit erhöhen, indem Sie diese Methode überschreiben und einfach *False* zurückgeben.

ProcessAccels `function ProcessAccels(var Message: TMsg): Boolean; virtual;`

Übernimmt die Bearbeitung der Accelerator-Botschaften. Wenn die Fenster Ihrer Anwendung auf Accelerator-Tasten nicht reagieren, können Sie die Leistungsfähigkeit erhöhen, indem Sie diese Methode überschreiben und einfach *False* zurückgeben.

ProcessMDI Accels

```
function ProcessMDIAccels(var Message: TMsg): Boolean;
virtual;
```

Übernimmt die Bearbeitung der MDI-Accelerator-Botschaften. Wenn Ihre Anwendung keine MDI-Anwendung ist, können Sie die Leistungsfähigkeit erhöhen, indem Sie diese Methode überschreiben und einfach *False* zurückgeben.

Run

```
procedure Run; virtual;
```

Startet die Anwendung durch Aufruf von *MessageLoop*, wenn die Initialisierung erfolgreich war (das heißt, das Feld *Status* ist 0).

Querverweis: *TApplication.MessageLoop*

SetKBHandler

```
procedure SetKBHandler(AWindowsObject: PWindowsObject);
```

Aktiviert die Tastaturverarbeitung für das angegebene Fenster durch Setzen von *KBHandlerWnd* auf *AWindowsObject.*

Querverweis*: TApplication.KBHandlerWnd*

ValidWindow

```
function ValidWindow(AWindowsObject: PWindowsObject):
PWindowsObject; virtual;
```

Bestimmt, ob *AWindowsObject* ein gültiges Objekt ist. In diesem Feld liefert *ValidWindow* einen Zeiger darauf zurück, andernfalls **nil**.

AWindowsObject ist ungültig, wenn eine der zwei Bedingungen erfüllt ist: Die Reservierung des Objekts berührte den Sicherheitsbereich (*LowMemory* ist *True*), oder das Feld *Status* von *AWindowsObject* ist ungleich 0.

Querverweis: *LowMemory,TWindowsObject.Status*

TBufStream

WObjects

Beschreibung

TBufStream implementiert eine gepufferte Version von *TDosStream.* Die zusätzlichen Felder geben die Größe und die Position des Puffers zusammen mit der aktuellen und der letzten Position innerhalb des Puffers an. Zusätzlich zum Überschreiben der acht Methoden von *TDosStream* definiert *TBufStream* die abstrakte Methode *TStream.Flush.* Der Konstruktor von *TBufStream* erzeugt und öffnet durch Aufruf von *TDosStream.Init* eine bezeichnete Datei und erstellt dann mit *GetMem* den Puffer.

TBufStream ist deutlich effizienter als *TDosStream,* wenn eine große Zahl Datenübertragungen geringer Datenmengen über den Stream stattfindet, wie zum Beispiel, wenn Objekte mittels *TStream.Get* und *TStream.Put* geladen und gespeichert werden.

Felder

Buffer
```
Buffer: Pointer;
```
Ein Zeiger auf den Anfang des Puffers des Streams.

BufSize
```
BufSize: Word;
```
Die Größe des Puffers in Byte.

BufPtr
```
BufPtr: Word;
```
Ein Offset vom Zeiger *Buffer*, der die aktuelle Position innerhalb des Puffers anzeigt.

BufEnd
```
BufEnd: Word;
```
Wenn der Puffer nicht voll ist, gibt *BufEnd* einen Offset vom Zeiger *Buffer* auf das letzte benutzte Byte im Puffer an.

Methoden

Init
```
constructor Init(FileName: PChar; Mode, Size: Word);
```
Erzeugt und öffnet die benannte Datei im Zugriffsmodus *Mode* durch Aufruf von *TDosStream.Init* und erzeugt mit einem Aufruf von *GetMem* einen Puffer von *Size* Byte. Die Felder *Handle*, *Buffer* und *BufSize* werden passend initialisiert.

Typische Puffergrößen liegen im Bereich von 512 Byte bis 2048 Byte.

Querverweis: *TDosStream.Init*

Done
```
destructor Done; virtual;
```
Schließt den Datei-Stream und gibt ihn frei; entleert den Puffer und gibt ihn frei.

Querverweis: *TBufStream.Flush*

Flush
```
procedure Flush; virtual;
```
Entleert den Puffer des aufrufenden Datei-Streams, vorausgesetzt der Stream ist *stOk*.

Querverweis: *TBufStream.Done*

GetPos
```
function GetPos: LongInt; virtual;
```
Gibt den Wert der aktuellen Position des aufrufenden Streams zurück (dies sollte nicht verwechselt werden mit *BufPtr*, der aktuellen Position innerhalb des Puffers).

Querverweis: *TBufStream.Seek*

GetSize
```
function GetSize: LongInt; virtual;
```
Entleert den Puffer und gibt die gesamte Größe des aufrufenden Streams in Byte zurück.

Read

```
procedure Read(var Buf; Count: Word); virtual;
```

Wenn der Stream *stOk* ist, liest diese Methode *Count* Bytes in den Puffer *Buf*, beginnend an der aktuellen Position des aufrufenden Streams.

Buf ist nicht der Puffer des Streams, sondern ein externer Puffer, der die vom Stream eingelesenen Daten aufnehmen soll.

Querverweis: *TBufStream.Write, stReadError*

Seek

```
procedure Seek(Pos: LongInt); virtual;
```

Entleert den Puffer und setzt anschließend die aktuelle Position auf *Pos* Bytes vom Anfang des aufrufenden Streams. Der Anfang eines Streams ist die Position 0.

Querverweis: *TBufStream.GetPos*

Truncate

```
procedure Truncate; virtual;
```

Entleert den Puffer und löscht anschließend alle Daten von der aktuellen Position bis zum Ende des aufrufenden Streams. Die aktuelle Position wird auf das neue Ende des Streams gesetzt.

Querverweis: *TBufStream.GetPos, TBufStream.Seek*

Write

```
procedure Write(var Buf; Count: Word); virtual;
```

Wenn der Stream *stOk* ist, schreibt diese Prozedur *Count* Bytes aus dem Puffer in den aufrufenden Stream, beginnend an der aktuellen Position.

Buf ist nicht der Puffer des Streams, sondern ein externer Puffer, der die Daten enthält, die in den Stream geschrieben werden. Wenn *Write* aufgerufen wird, zeigt *Buf* auf die Variable, deren Wert geschrieben wird.

Querverweis: *TBufStream.Read, stWriteError*

TButton

WObjects

Beschreibung

TButton ist ein Schnittstellenobjekt, das einen Aktionsschalter in Windows repräsentiert. *TButton*-Objekte werden normalerweise nicht in Dialogfenstern (*TDialog* oder *TDlgWindow*) benutzt. Sie werden eingesetzt, wenn man einen einzelnen Aktionsschalter im Client-Bereich eines anderen Fensters darstellen will.

Es gibt zwei Typen von Aktionsschaltern: Ein regulärer Aktionsschalter erscheint mit einer dünnen Umrandung, ein Standardschalter erscheint mit einer dicken Umrandung und steht für die Standardaktion des Fensters. In einem Fenster gibt es höchstens einen Standardschalter.

Methoden

Init

```
constructor Init(AParent: PWindowsObject; AnId:
Integer; AText: PChar; X, Y, W, H: Integer; IsDefault:
Boolean);
```

Initialisiert ein Schalterobjekt für das übergebene übergeordnete Fenster (*AParent*) mit der Steuerelement-ID (*AnId*), dem zugehörigen Text (*AText*), der relativen Position zum Ursprung des Client-Bereichs des übergeordneten Fensters (*X,Y*), der Breite (*W*) und der Höhe (*H*). Ruft *TControl.Init* auf und fügt *bs_DefPushButton* zum Feld *Attr.Style* hinzu, wenn *IsDefault True* ist, ansonsten *bs_PushButton*.

Querverweis: *TControl.Init*

InitResource

```
constructor InitResource(AParent: PWindowsObject,
ResourceID: Word);
```

Ordnet den Aktonsschalter durch Konstruktion eines Object-Windows-Objektes einem entsprechenden Schalterelement zu, das von einer Dialogressource definiert wurde.

Ruft *TControl.InitResource* und *DisableTransfer* auf, um den Schalter vom Transfermechanismus auszuschließen, da er keine Daten transferieren muß.

Querverweis: *TControl.InitResource, TWindowsObject.DisableTransfer*

GetClassName

```
function GetClassName: PChar; virtual;
```

Gibt die Bezeichnung der Fensterklasse von *TButton* zurück: 'Button'.

TCheckBox **WObjects**

Beschreibung

TCheckBox ist ein Schnittstellenobjekt, das einen Auswahlschalter repräsentiert. *TCheckBox*-Objekte werden normalerweise nicht in Dialogfenstern (*TDialog* oder *TDlgWindow*) verwendet, sondern werden eingesetzt, wenn man ein selbständiges Auswahlfeld im Client-Bereich eines anderen Fensters darstellen will.

Auswahlfelder haben zwei oder drei Zustände: »markiert«, »nicht markiert« und gegebenenfalls »nicht aktivierbar«. *TCheckBox*-Methoden sind hauptsächlich für die Verwaltung des Status des Auswahlfeldes zuständig. Optional kann ein Auswahlfeld Teil einer Gruppe sein (*TGroupBox*), die ihre Steuerelemente visuell und funktionell gruppiert.

Felder

Group

```
Group: PGroupBox;
```

Group zeigt auf das *TGroupBox*-Steuerelementobjekt, das das Auswahlfeld mit anderen Auswahlfeldern und Schaltfeldern (*TRadioButton*) vereinigt.

Wenn das Auswahlfeld nicht Teil einer Gruppe ist, ist *Group* gleich **nil**.

Querverweis: *TGroupBox, TRadioButton*

Methoden

Init

```
constructor Init(AParent: PWindowsObject; AnID:
Integer; ATitle: PChar; X, Y, W, H: Integer; AGroup:
PGroupBox);
```

Initialisiert ein Auswahlfeldobjekt für das übergebene übergeordnete Fenster (*AParent*) mit der Steuerelement-ID (*AnId*), dem zugehörigen Text (*ATitle*), der Position relativ zum Ursprung des Client-Bereichs des übergeordneten Fensters (*X,Y*), der Breite (*W*), der Höhe (*H*) und dem zugehörigen Gruppenfeld (*AGroup*). *TCheckBox.Init* setzt das Feld *Attr.Style* des Auswahlfeldes auf *ws_Child* **or** *ws_Visible* **or** *ws_TabStop* **or** *bs_AutoCheckBox*.

InitResource

```
constructor InitResource(AParent: PWindowsObject;
ResourceID: Word);
```

Diese Methode ordnet einer durch *ResourceID* bezeichneten Ressource ein *TCheckBox*-Objekt zu, indem sie *TButton.Init-Resource* aufruft und dann den Transfermechanismus durch Aufruf von *EnableTransfer* einschaltet.

Load

```
constructor Load(var S: TStream);
```

Diese Methode initialisiert und lädt ein Auswahlfeld aus dem Stream *S*, indem sie zuerst *TButton.Load* aufruft und dann das von *TCheckBox* hinzugefügte Feld (*Group*) liest.

Querverweis: *TWindow.Load*

BNClicked

```
procedure BNClicked(var Msg: TMessage); virtual
nf_First + bn_Clicked;
```

Reagiert auf Benachrichtigungsbotschaften, die anzeigen, daß das Auswahlfeld angeklickt wurde, mit der Änderung dessen Zustandes.

Ist das Feld *Group* des Auswahlfeldes nicht mit **nil** belegt, benachrichtigt *TCheckBox.BNClicked* dieses *TGroupBox*-Objekt durch Aufruf seiner Methode *SelectionChanged*.

Querverweis: *TGroupBox.SelectionChanged*

Check

```
procedure Check;
```

Versetzt das Auswahlfeld durch Aufruf von *SetCheck* in den markierten Zustand.

Querverweis: *TCheckBox.SetCheck*

GetCheck

```
function GetCheck: Word;
```

Gibt *bf_Unchecked* (0) zurück, wenn das Auswahlfeld nicht markiert ist, *bf_Checked* (1), wenn es markiert ist oder *bf_Grayed* (2), wenn es nicht aktivierbar ist.

SetCheck

```
procedure SetCheck(CheckFlag: Word);
```

Versetzt das Auswahlfeld in den von *CheckFlag* angegebenen Zustand. 0 für *CheckFlag* bedeutet »nicht markiert«, 1 »markiert« und 2 »nicht aktivierbar«. *SetCheck* informiert auch die Gruppe des Auswahlfeldes, daß sich dessen Zustand geändert hat.

Querverweis: *TGroupBox.SelectionChanged*

Store

```
procedure Store(var S: TStream);
```

Die Methode speichert das Auswahlfeld im Stream *S*, indem sie zuerst *TButton.Store* aufruft und dann das von *TCheckBox* hinzugefügte Feld (*Group*) beschreibt.

Querverweis: *TWindow.Store*

Toggle

```
procedure Toggle;
```

Diese Methode schaltet den Zustand des Auswahlfensters durch Aufruf von *Check* oder *Uncheck* um. Bei einem Auswahlfeld mit drei Schaltzuständen durchläuft sie alle drei Zustände: »markiert«, »nicht markiert« und »nicht aktivierbar«.

Querverweis: *TCheckBox.Check, TCheckBox.Uncheck*

Transfer

```
function Transfer(DataPtr: Pointer; TransferFlag:
Word): Word; virtual;
```

Transferiert den Zustand des Auswahlfeldes in einem Wert vom Typ Word (*bf_Checked* für »markiert« und *bf_Unchecked* für »nicht markiert«) an die oder von der Speicherposition, auf die *DataPtr* zeigt. Ist *TransferFlag tf_GetData*, wird der Zustand des Auswahlfeldes an die Speicherposition transferiert. Ist *TransferFlag tf_SetData*, wird das Auswahlfeld in den Zustand versetzt, der an der Speicherposition angegeben ist. *Transfer* gibt die Anzahl der Bytes zurück, die an der Speicherposition gespeichert oder von dort abgerufen wurden. Wenn Sie *tf_SizeData* übergeben, gibt *Transfer* die Größe der Transferdaten zurück (zwei Byte).

Uncheck `procedure Uncheck;`

Versetzt das Auswahlfeld durch Aufruf von *SetCheck* in den Status »nicht markiert«.

Querverweis: *TCheckBox.SetCheck*

TCollection **WObjects**

Beschreibung *TCollection* ist ein abstrakter Typ zur Implementierung einer beliebigen Kollektion von Elementen, einschließlich anderer Objekte. *TCollection* ist allgemeiner als die bekannten Datenstrukturen Array, Liste oder Menge. *TCollection*-Objekte können sich während der Laufzeit selbständig und dynamisch vergrößern und bieten einen Basistyp für viele spezialisierte Typen wie etwa *TSortedCollection* und *TStrCollection*. Zusätzlich zu Methoden, die dem Hinzufügen und Löschen von Elementen dienen, bietet *TCollection* mehrere Iterator-Routinen, die für jedes Element einer Kollektion eine Prozedur oder Funktion aufrufen.

Felder

Items `Items: PItemList;`

Ein Zeiger auf ein Array von Elementzeigern.

Querverweis: *TItemList*-Typ

Count `Count: Integer;`

Die aktuelle Anzahl von Einträgen in der Kollektion, bis hin zu *MaxCollectionSize*.

Querverweis: Variable *MaxCollectionSize*

Limit `Limit: Integer;`

Die aktuelle reservierte Größe der *Items*-Liste (in Elementen).

Querverweis: *Delta,TCollection.Init*

Delta `Delta: Integer;`

Die Anzahl der Elemente, um die die *Items*-Liste anwachsen soll, sobald sie voll wird. Wenn *Delta* 0 ist, kann die Kollektion nicht über die von *Limit* gesetzte Grenze hinaus anwachsen.

Hinweis: Das Anwachsen der Größe einer Kollektion geht auf Kosten der Leistungsfähigkeit. Die Werte von *Limit* und *Delta* sollten daher mit Bedacht gewählt werden, um die Anzahl der benötigten Speicherplatzreservierungen zu minimieren.

Querverweis: *Limit, TCollection.Init.*

Methoden

Init

```
constructor Init(ALimit, ADelta: Integer);
```

Erzeugt eine Kollektion, wobei *Limit* auf *ALimit* und *Delta* auf *ADelta* gesetzt sind.

Die anfängliche Anzahl der Elemente wird durch *ALimit* begrenzt, die Kollektion kann aber mit der Schrittweite *ADelta* solange wachsen, bis der Speicher erschöpft ist oder die Anzahl der Elemente *MaxCollectionSize* erreicht.

Querverweis: *TCollection.Limit, TCollection.Delta*

Load

```
constructor Load(var S: TStream);
```

Erzeugt und lädt eine Kollektion aus dem angegebenen Stream. *TCollection.Load* ruft *GetItem* für jedes Element der Kollektion auf.

Querverweis: *TCollection.GetItem*

Done

```
destructor Done; virtual;
```

Löscht alle Elemente in der Kollektion durch Freigabe des Speicherplatzes mit *TCollection.FreeAll* und das Setzen von *Limit* auf 0.

Querverweis: *TCollection.FreeAll, TCollection.Init*

At

```
function At (Index: Integer): Pointer;
```

Gibt einen Zeiger auf das mit *Index* indizierte Element in der Kollektion zurück. Mit dieser Methode kann eine Kollektion wie ein indiziertes Array behandelt werden. Wenn *Index* kleiner als 0 oder größer als bzw. gleich *Count* ist, wird die Methode *Error* mit dem Argument *coIndexError* für *Code* aufgerufen und ein Wert von **nil** zurückgegeben.

Querverweis: *TCollection.IndexOf*

AtDelete

```
procedure AtDelete(Index: Integer);
```

Löscht das Element an der Position *Index* und verschiebt die folgenden Elemente um eine Position nach oben. *Count* wird um 1 dekrementiert, der durch *Limit* für die Kollektion reservierte Speicher wird jedoch nicht verkleinert. Wenn *Index* kleiner als 0 oder größer als bzw. gleich *Count* ist, wird die Methode *Error* mit dem Argument *coIndexError* für *Code* aufgerufen.

Querverweis: *TCollection.FreeItem, TCollection.Free, TCollection.Delete*

AtFree

```
procedure AtFree(Index: Integer);
```

Löscht das Element an der Position *Index* und gibt es frei.

AtInsert

```
procedure AtInsert(Index: Integer; Item: Pointer);
```

Fügt *Item* an der Position *Index* ein und verschiebt die folgenden Elemente um eine Position nach unten. Wenn *Index* kleiner als 0 oder größer als *Count* ist, wird die *Error*-Methode mit dem Argument *coIndexError* für *Code* aufgerufen und das neue Element wird nicht eingefügt. Ist *Count* vor dem Aufruf von *AtInsert* gleich *Limit*, wird mit einem Aufruf von *SetLimit* die der Kollektion zugewiesen Größe um *Delta* Elemente erweitert. Schlägt der Aufruf von *SetLimit* zum Erweitern der Kollektion fehl, wird die *Error*-Methode mit dem Argument *coOverflow* für *Code* aufgerufen und das neue Element wird nicht eingefügt.

Querverweis: *TCollection.At, TCollection.AtPut*

AtPut

```
procedure AtPut(Index: Integer; Item: Pointer);
```

Ersetzt das Element an der Indexposition *Index* durch das von *Item* vorgegebene Element.

Wenn *Index* kleiner als 0 oder größer als bzw. gleich *Count* ist, wird die *Error*-Methode mit dem Argument *coIndexError* für *Code* aufgerufen.

Querverweis: *TCollection.A*t, *TCollection.AtInsert*

Delete

```
procedure Delete(Item: Pointer),
```

Löscht das durch *Item* vorgegebene Element aus der Kollektion und ist äquivalent mit *AtDelete(IndexOf(Item))*.

Querverweis: *TCollection.Delete, TCollection.AtDelete*

Error

```
procedure Error(Code, Info: Integer); virtual;
```

Diese Methode wird immer dann aufgerufen, wenn ein Kollektionsfehler erkannt wird. Per Voreinstellung produziert sie einen Laufzeitfehler mit dem Code (212 - *Code*).

Querverweis: *coXXXX*- Kollektionskonstanten

DeleteAll

```
procedure DeleteAll;
```

Setzt *Count* auf 0 und löscht dadurch alle Elemente aus der Kollektion.

Querverweis: *TCollection.Delete, TCollection.AtDelete*

FirstThat

```
function FirstThat(Test: Pointer): Pointer;
```

FirstThat wendet auf jedes Element in der Kollektion eine mit dem Funktionszeiger *Test* bezeichnete Boolesche Funktion an, bis *Test True* zurückgibt. Das Ergebnis ist der Elementzeiger, für den *Test True* zurückgibt, oder **nil**, wenn die Funktion *Test* für alle Elemente *False* zurückgibt. *Test* muß auf eine lokale als **far** deklarierte Funktion zeigen, die einen Parameter vom Typ

Pointer entgegennimmt und einen booleschen Wert zurückgibt. Beispiel:

```
function  Matches(Item: Pointer): Boolean;       far ;
```

Die Funktion *Test* darf *keine* globale Funktion sein. Angenommen, *List* ist ein *TCollection*-Objekt, dann entspricht die Anweisung

```
P := List.FirstThat(@Matches);
```

diesem Programmfragment:

```
I := 0;
while  (I < List.Count)    and not  Matches(List.At(I))
       do  Inc(I) ;
if I < List.Count   then  P := List.At(I)   else  P :=  nil ;
```

Querverweis: *TCollection.LastThat, TCollection.ForEach*

ForEach

```
procedure ForEach(Action: Pointer);
```

ForEach führt mit jedem Element in der Kollektion eine Aktion aus, auf die der Prozedurzeiger *Action* zeigt. *Action* muß auf eine lokale, als **far** deklarierte Prozedur zeigen, die einen Parameter vom Typ *Pointer* entgegennimmt. Beispiel:

```
function  PrintItem(Item: Pointer);      far ;
```

Die Prozedur *Action* darf *keine* globale Prozedur sein. Angenommen, *List* ist ein *TCollection*-Objekt, dann entspricht die Anweisung

```
List.ForEach(@PrintItem);
```

diesem Programmfragment:

```
for  I := 0  to  List.Count - 1   do  PrintItem(List.At(I));
```

Querverweis: *TCollection.FirstThat, TCollection.LastThat*

Free

```
procedure Free(Item: Pointer);
```

Löscht das gegebene *Item* und gibt es frei.

```
Free(Item);
```

ist äquivalent zu

```
Delete(Item);
FreeItem(Item);
```

Querverweis: *TCollection.FreeItem, TCollection.Delete*

FreeAll

```
procedure FreeAll;
```

Löscht alle Elemente in der Kollektion und gibt sie frei.

Querverweis: *TCollection.DeleteAll*

FreeItem

```
procedure FreeItem(Item: Pointer); virtual;
```

Die Methode *FreeItem* gibt das angegebene *Item* frei. Per Voreinstellung nimmt *TCollection.FreeItem* an, daß *Item* ein Zeiger auf den Nachkommen eines *TObject* ist und ruft daher den Destruktor *Done* auf:

```
if Item <> nil then Dispose(Pobject(Item), Done);
```

FreeItem wird von *Free* und *FreeAll* aufgerufen und sollte niemals direkt aufgerufen werden.

Querverweis: *TCollection.Free, TCollection.FreeAll*

GetItem

```
function GetItem(var S:TStream): Pointer; virtual;
```

Wird von *TCollection.Load* für jedes Element in der Kollektion aufgerufen. Diese Methode kann überschrieben, sollte aber nicht direkt aufgerufen werden. Per Voreinstellung nimmt *TCollection.GetItem* an, daß die Elemente in der Kollektion Nachkommen von *TObject* sind und ruft daher *TStream.Get* auf, um das Element zu laden:

```
GetItem := S.Get;
```

Querverweis: *TStream.Get, TCollection.Load, TCollection.Store*

IndexOf

```
function IndexOf(Item: Pointer): Integer; virtual;
```

Gibt den Index des angegebenen *Item* zurück und ist das Gegenstück zu *TCollection.At*. Wenn *Item* sich nicht in der Kollektion befindet, gibt *IndexOf* -1 zurück.

Querverweis: *TCollection.At*

Insert

```
procedure Insert(Item: Pointer); virtual;
```

Fügt das Element *Item* in die Kollektion ein und gleicht andere Indizes ab, wenn dies erforderlich ist. Per Voreinstellung werden Einfügungen durch Aufruf von *AtInsert(Count, Item)* am Ende der Kollektion vorgenommen.

Querverweis: *TCollection.AtInsert*

LastThat

```
function LastThat(Test: Pointer): Pointer;
```

LastThat wendet auf jedes Element in der Kollektion in umgekehrter Reihenfolge eine durch den Funktionszeiger *Test* vorgegebene boolesche Funktion an, bis *Test True* zurückgibt. Das Ergebnis ist der Elementzeiger, für den *Test True* zurückgegeben hat oder **nil**, wenn die Funktion *Test false* für alle Elemente zurückgegeben hat. *Test* muß auf eine lokale als **far** deklarierte Funktion zeigen, die einen Parameter vom Typ Pointer entgegennimmt und einen booleschen Wert zurückliefert. Beispiel:

```
function  Matches(Item: Pointer): Boolean;      far ;
```

Die Funktion *Test* darf *keine* globale Funktion sein. Angenommen, *List* ist ein *TCollection*-Objekt, dann entspricht die Anweisung

```
P := List.LastThat(@Matches);
```

diesem Programmfragment:

```
I := List.Count - 1;
while (I >= 0) and not Matches(List.At(I)) do Dec(I);
if I >= 0 then P := List.At(I) else P := nil;
```

Querverweis: *TCollection, FirstThat, TCollection.ForEach*

Pack

```
procedure Pack;
```

Löscht alle **nil**-Zeiger in der Kollektion.

Querverweis: *TCollection.Delete, TCollection.DeleteAll*

PutItem

```
procedure PutItem(var S: TStream; Item: Pointer);
virtual;
```

Wird von *TCollection.Store* für jedes Element in der Kollektion aufgerufen. Diese Methode kann überschrieben, sollte aber nicht direkt aufgerufen werden. Per Voreinstellung nimmt *TCollection.PutItem* an, daß die Elemente in der Kollektion Nachkommen von *TObject* sind und ruft daher *TStream.Put* auf, um das Element zu speichern:

```
S.Put(Item);
```

Querverweis: *TCollection.GetItem, TCollection.Store, TCollection.Load*

SetLimit

```
procedure SetLimit(ALimit: Integer); virtual;
```

Erweitert oder verkleinert die Kollektion durch Ändern der zugewiesenen Größe auf *ALimit*. Ist *ALimit* kleiner als *Count*, wird sie auf *Count*, ist *ALimit* größer als *MaxCollectionSize*, wird sie auf *MaxCollectionSize* gesetzt. Wenn sich dann *ALimit* vom aktuellen *Limit* unterscheidet, wird ein neues *Items*-Array zugewiesen, das alte *Items*-Array in das neue Array kopiert und das alte Array freigegeben.

Querverweis: *TCollection.Limit, TCollection.Count, MaxCollectionSize*-Variable

Store

```
procedure Store(var S: TStream);
```

Speichert die Kollektion und alle ihre Elemente im Stream *S*. *TCollection.Store* ruft für jedes Element in der Kollektion *TCollection.PutItem* auf.

Querverweis: *TCollection.PutItem*

TComboBox **WObjects**

Beschreibung

TComboBox ist ein Schnittstellenobjekt, das ein entsprechendes Kombinationsfenster repräsentiert. *TComboBox*Objekte werden normalerweise nicht in Dialogfenstern (*TDialog* oder *TDlgWindow*) verwendet, sondern werden eingesetzt, wenn man ein selbständiges Kombinationsfenster im Client-Bereich eines anderen Fensters darstellen möchte. Kombinationsfensterobjekte erben das meiste ihrer Funktionalität von *TListBox*.

Es gibt drei Typen von Kombinationsfenstern, die den folgenden Windows-Stilkonstanten entsprechen: *cbs_Simple, cbs_DropDown* und *cbs_DropDownList*. Diese werden an den Konstruktor *Init* übergeben, der Windows wiederum mitteilt, welcher Typ von Kombinationsfensterelement erzeugt werden soll.

Felder

TextLen

```
TextLen: Word;
```

TextLen enthält die Länge des Zeichenpuffers im Editfeld des Kombinationsfensters, die auch die Anzahl der von *Transfer* transferierten Bytes ist. *TextLen* wird von *Init* gesetzt.

Methoden

Init

```
constructor Init(AParent: PWindowsObject; AnID:
Integer; X, Y, W, H: Integer; AStyle, ATextLen: Word);
```

Initialisiert durch den Aufruf von *TListBox.Init* ein Kombinationsfensterobjekt für das übergebene übergeordnete Fenster (*AParent*) mit der Steuerelement-ID (*AnID*), der relativen Position zum Ursprung des Client-Bereichs des übergeordneten Fensters (*X, Y*), der Breite (*W*) und der Höhe (*H*). Diese Methode setzt *TextLen* auf *ATextLen, Attr.Style* auf (*AStyle* **and not** *lbs_Notify*) **or** *AStyle* **or** *cbs_Sort* **or** *ws_VScroll* **or** *ws_HScroll*.

Querverweis: *TListBox.Init, cbs_*-Kombinationsfensterkonstanten

InitResource

```
constructor InitResource(AParent: PWindowsObject;
ResourceID: Integer; ATextLen: Word);
```

Ordnet einem *TComboBox*-Objekt die über *ResourceID* bezeichnete Ressource mit einer maximalen Länge von *ATextLen* - 1 zu.

Load

```
constructor Load(var S: TStream);
```

Initialisiert und lädt ein Kombinationsfenster aus dem Stream *S*, indem die Methode zuerst *TListBox.Load* aufruft und dann das von *TComboBox* hinzugefügte Feld liest (*TextLen*).

Querverweis: *TWindow.Load*

GetClassName `function GetClassName: PChar; virtual;`

Gibt 'ComboBox', die Bezeichnung der Fensterklasse von *TComboBox* zurück.

HideList `procedure HideList;`

Bewirkt das Ausblenden der Liste aus einem damit ausgestatteten Kombinationsfenster.

ShowList `procedure ShowList;`

Bewirkt das Einblenden der Liste eines damit ausgestatteten Kombinationsfensters.

Store `procedure Store(var S: TStream);`

Diese Methode speichert das Kombinationsfenster im Stream *S*, indem sie zuerst *TListBox.Store* aufruft und dann das von *TComboBox* hinzugefügte Feld (*Style*) beschreibt.

Querverweis: *TWindow.Store*

Transfer `function Transfer(DataPtr: Pointer; TransferFlag: Word); virtual;`

Transferiert Daten zu oder von dem Record, auf den *DataPtr* zeigt. Das erste Feld dieses Records muß ein Zeiger auf eine Stringkollektion (*PStrCollection*)sein, die die Einträge der Liste des Kombinationsfensters enthält. Das zweite Feld muß ein Array aus Zeichen sein, in das der aktuell ausgewählte Eintrag transferiert wird. Der Transferpuffer könnte so aussehen:

```
type
  TComboXferRec =    record
    Strings: PStrCollection;
    Selection:    array [0..TextLen-1]    of Char;
   end ;
```

wobei *TextLen* durch den im Konstruktor *Init* übergebenen Wert ersetzt werden müßte.

Ist *TransferFlag tf_SetData*, werden die Daten dieses Records in das Kombinationsfenster transferiert. Ist *TransferFlag tf_GetData*, werden die Daten des Kombinationsfensters an den Record *DataPtr* transferiert. In diesen Fällen gibt *Transfer* die Größe der transferierten Daten zurück. Ist *TransferFlag tf_SizeData*, gibt *Transfer* die Größe der Transferdaten zurück.

Querverweis: *TListBox.Transfer*

TControl **WObjects**

Beschreibung *TControl* ist ein abstrakter Objekttyp, der der Vorfahre aller Typen von Steuerelementobjekten wie *TScrollBar* und *TButton*

ist sowie von *TMDIClient*, einem spezialisierten Steuerelement für MDI-Anwendungen. Steuerelementobjekte werden im allgemeinen nicht in Dialogfenstern (*TDialog* und *TDlgWindows*) verwendet, sondern nur als selbständige Steuerelemente im Client-Bereich anderer Fenster.

Methoden

Init

```
constructor Init(AParent: PWindowsObject; AnId:
Integer; ATitle: PChar; X, Y, W, H: Integer);
```

Initialisiert ein Steuerelementobjekt für das übergebene übergeordnete Fenster (*AParent*) mit der Steuerelement-ID (*AnID*), dem zugeordneten Text (*ATitle*), der Position relativ zum Ursprung des Client-Bereichs des übergeordneten Fensters (*X,Y*), der Breite (*W*) und der Höhe (*H*). Mit diesen Argumenten füllt diese Methode das von *TWindow* geerbte Feld *Attr* des Steuerelements. Per Voreinstellung setzt sie *Attr.Style* auf *ws_Child* **or** ws_Visible **or** *ws_Group* **or** *ws_TabStop*, so daß alle Steuerelementobjekte sichtbare untergeordnete Fenster sind.

InitResource

```
constructor InitResource(AParent: PWindowsObject;
ResourceID: Word);
```

Diese Methode ordnet einem Steuerelementobjekt das Steuerelement in der von *ResourceID* bezeichneten Ressource zu und ruft *TWindow.InitResource* und *EnableTransfer* auf, um den Transfermechanismus einzuschalten.

Querverweis: *TWindow.InitResource, TWindowsObject.EnableTransfer*

GetClassName

```
function GetClassName: PChar; virtual;
```

Abstrakte Methode zur Überschreibung durch Nachkommen des Objekts.

Register

```
function Register: Boolean; virtual;
```

Gibt lediglich *True* zurück, um anzuzeigen, daß Nachkommen von *TControl* vordefinierte Fensterklassen benutzen.

WMPaint

```
procedure WMPaint(var Msg: TMessage); virtual wm_First
+ wm_Paint;
```

Ruft zum standardmäßigen Neuzeichnen von Steuerelementobjekten *DefWndProc* auf.

Querverweis: *TWindow.DefWndProc*

TDialog

WObjects

Beschreibung *TDialog* definiert Objekte, die als modale und nichtmodale Dialogfenster in Windows-Anwendungen benutzt werden. Ein

TDialog-Objekt verfügt über eine zugeordnete Dialogressource, die das Erscheinungsbild und die Anordnung seiner Steuerelemente beschreibt. Diese Ressource wird im Aufruf von *TDialog.Init* angegeben.

Dialogfensterobjekte können durch Aufrufe ihrer Methoden *Execute* bzw. *Create* entweder modalen oder nichtmodalen Dialogelementen zugeordnet werden. Normalerweise sollten Dialogobjekte jedoch über die *TApplication*-Methoden *ExecDialog* und *MakeWindow* aktiviert werden, da diese vor dem Aufruf von *Execute* und *Create* auf Speichermangel prüfen.

Beachten Sie, daß die Erzeugung eines modalen Dialogs die weiteren Operationen seines übergeordneten Fensters verhindert.

Felder

Attr

```
Attr: TDialogAttr;
```

Attr enthält die Erzeugungsattribute des Dialogfensters.

TDialogAttr ist folgendermaßen definiert:

```
TDialogAttr =   record
  Name: PChar;
  Param: LongInt;
end ;
```

Das Feld *Name* enthält die Bezeichnung oder die ID der Dialogressource. Das Feld *Param* enthält einen Parameter, der an die Dialogprozedur übergeben wird, wenn der Dialog erzeugt wird.

IsModal

```
IsModal: Boolean;
```

IsModal ist *True*, wenn der Dialog modal ist und *False*, wenn er nichtmodal ist.

Methoden

Init

```
constructor Init(AParent: PWindowsObject; AName: PChar);
```

Initialisiert das Dialogobjekt durch Aufruf von *TWindowsObject.Init* und Übergabe des übergeordneten Fensters *AParent*. Anschließend setzt *TDialog.Init* das Feld *Attr.Name* auf den in *AName* übergebenen String.

Der String kann die symbolische Bezeichnung der Dialogressource sein (z.B. 'MITARBEITERINFO') oder eine in den Typ *PChar* umgewandelte Integer-ID (z.B. *PChar(120)*). *Init* ruft auch *DisableAutoCreate* auf, so daß Dialoge nicht automatisch erzeugt und mit ihren übergeordneten Fenstern dargestellt werden.

Querverweis: *TWindowsObject.Init, TWindowsObject.DisableAutoCreate*

Load
```
constructor Load(var S: TStream);
```
Diese Methode initialisiert und lädt ein Dialogfenster aus dem Stream *S*, indem sie *TWindowsObject.Load* aufruft und dann die von *TDialog* hinzugefügten Felder (*Attr* und *IsModal*) liest.

Querverweis: *TWindowsObject.Load*

Done
```
destructor Done; virtual;
```
Gibt das Dialogfensterobjekt durch Aufruf von *TWindows-Object.Done* frei.

Querverweis: *TWindowsObject.Done*

Cancel
```
procedure Cancel(var Msg: TMessage); virtual id_First +
id_Cancel;
```
Beantwortet die Betätigung des Schalters »Cancel« bzw »Abbrechen« des Dialogs mit einem Aufruf von *EndDlg* mit dem Wert *id_Cancel*.

Querverweis: *TDialog.EndDlg*

Create
```
function Create: Boolean; virtual;
```
Erzeugt das einem nichtmodalen Dialogobjekt entsprechende Dialogelement: *TDialog.Create* gibt *True* zurück, wenn sie erfolgreich ist, andernfalls ruft sie *Error* mit dem Fehlercode *em_InvalidWindow* auf.

Querverweis: *TDialog.Execute, TWindowsObject.Error*

DefWndProc
```
procedure DefWndProc(var Msg: TMessage); virtual;
```
Löst die Standardbotschaftsverarbeitung von Windows aus, indem sie das Ergebnis der übergebenen Botschaft auf 0 setzt.

Destroy
```
procedure Destroy; virtual;
```
Diese Methode löscht das Dialogfenster und gibt es frei, indem sie *EndDlg* aufruft und die Konstante *id_Cancel* übergibt.

Querverweis: *TDialog.EndDlg*

EndDLg
```
procedure EndDlg(ARetValue: Integer); virtual;
```
Löscht modale und nichtmodale Dialogfenster. *ARetValue* wird als der Rückgabewert von *TDialog.Execute* für modale Dialogfenster zurückgeliefert.

Querverweis: *TDialog.Execute, TDialog.Create*

EnterCancel
```
procedure EnterCancel(var Msg: TMessage); virtual
cm_First + id_Cancel;
```
Beantwortet das Betätigen der Eingabetaste in einem Dialog mit einem Aufruf von *Cancel*. In diesem Fall war der Schalter

»Cancel« bzw »Abbrechen« der voreingestellte oder hervorgehobene Schalter.

Querverweis: *TDialog.Cancel*

EnterOk

```
procedure EnterOk(var Msg: TMessage); virtual cm_First
+ id_Ok;
```

Beantwortet das Drücken der Eingabetaste in einem Dialogfenster mit einem Aufruf von *Ok*. In diesem Fall war der Schalter »Ok« der voreingestellte oder hervorgehobene Schalter.

Querverweis: *TDialog.Ok*

Execute

```
function Execute: Integer; virtual;
```

Erzeugt das einem modalen Dialogobjekt entsprechende Dialogelement. Diese Methode arbeitet, solange das Dialogfenster auf dem Bildschirm ist, bis seine Methode *EndDlg* aufgerufen wird. Bevor die Methode zurückkehrt, setzt sie *HWindow* auf 0 zurück.

Bei erfolgreicher Ausführung gibt *TDialog.Execute* den von *TDialog.EndDlg* zurückgelieferten Integerwert zurück, andernfalls ruft *Execute Error* mit dem Fehlercode *em_InvalidWindow* auf.

Querverweis: *TDialog.EndDlg, TDialog.Create, TWindowsObject.Error*

GetItemHandle

```
function GetItemHandle(DlgItemID: Integer): HWnd;
```

Gibt das Handle des Steuerelements des Dialogs zurück, das von der übergebenen ID *DlgItemID* bezeichnet wird.

Ok

```
procedure Ok(var Msg: TMessage); virtual id_First +
id_Ok;
```

Beantwortet das Drücken des OK-Schalters des Dialogs mit dem Aufruf von *CanClose* und von *EndDlg* mit dem Wert *id_Ok*. Ebenso ruft sie *TransferData* (*tf_GetData*) auf, um die Daten der Steuerelemente in den Transferpuffer zu transferieren.

Querverweis: *TDialog.EndDlg*

SendDlgItemMsg

```
function SendDlgItemMsg(DlgItemID: Integer; AMsg,
WParam: Word; LParam: LongInt): LongInt;
```

Sendet eine von *AMsg* bezeichnete Steuerelementbotschaft an das Steuerelement des Dialogs, das durch die von *DlgItemID* übergebene ID bezeichnet wird. *WParam* und *LParam* werden die Parameter der Windows-Botschaft.

SendDlgItemMsg gibt den vom Steuerelement zurückgelieferten Wert bzw. 0 zurück, wenn die Steuerelement-ID ungültig ist.

Store

```
procedure Store(var S: TStream);
```

Diese Methode speichert das Dialogfeld im Stream *S* ab, indem sie zuerst *TWindowsObject.Store* aufruft und dann die von *TDialog* hinzugefügten Felder (*Attr* und *IsModal*) beschreibt.

Querverweis: *TWindowsObject.Store*

WMInitDialog

```
procedure WMInitDialog(var Msg: TMessage); virtual
wm_First + wm_InitDialog;
```

Die Methode *TDialog.WMInitDialog* wird automatisch aufgerufen, kurz bevor der Dialog dargestellt wird. Sie ruft *SetupWindow* auf, um die für den Dialog oder seine Steuerelemente notwendigen Initialisierungen vorzunehmen.

Querverweis: *TWindowsObject.SetupWindow*

TDlgWindow **WObjects**

Beschreibung

Von *TDlgWindow* definierte Dialogfenster kombinieren einige Merkmale von Dialogen und Fenstern. Wie ein Dialog verfügt ein Dialogfenster über eine zugeordnete Dialogressource, die das Erscheinungsbild und die Position seiner Steuerelemente beschreibt. Dagegen hat es wie ein Fenster eine Fensterklasse, die Symbole und Mauszeiger definieren kann. Um diese Dialogfenster zu erzeugen und darzustellen, benutzen Sie die nichtmodalen Dialogmethoden *Create* anstelle von *Execute*.

Methoden

Init

```
constructor Init(AParent: PWindowsObject; AName: PChar);
```

Initialisiert durch den Aufruf von *TDialog.Init* ein neues *TDlgWindow*-Object und ruft *EnableAutoCreate* auf, so daß es als untergeordnetes Fenster zusammen mit seinem übergeordneten Fenster erzeugt und dargestellt wird.

Querverweis: *TDialog.Init, TWindowsObject.EnableAutoCreate*

Cancel

```
procedure Cancel(var Msg: TMessage); virtual id_First +
id_Cancel;
```

Beantwortet das Anklicken des Schalters *Cancel* mit einem Aufruf von *CanClose*, um das Schließen des Fensters genehmigen zu lassen, und gibt anschließend das Dialogfenster frei.

Querverweis: *TDlgWindow.Ok*

Create

```
function Create: Boolean; virtual;
```

Registriert die Klasse des Dialogfensters und ruft *TDialog.Create* auf. *TDlgWindow.Create* gibt bei erfolgreicher Ausführung *True* zurück.

Querverweis: *TWindowsObject.Register, TDialog.Create*

GetWindowClass

```
procedure GetWindowClass(var AWndClass: TWndClass);
virtual;
```

Definiert den Record für die Standard-Fensterklasse und gibt ihn in *AWndClass* zurück. Diese Fensterklasse definiert ein Standardsymbol, einen Standardmauszeiger und kein Menü. Überschreiben Sie *GetWindowClass* ebenso wie *GetClassName* für Ihre *TDlgWindow*-Nachkommen. Stellen Sie jedoch sicher, daß Ihre *GetWindowClass*-Methode *TDlgWindow.GetWindowClass* aufruft, bevor sie irgendwelche *TWndClass*-Felder verändert.

Ok

```
procedure Ok(var Msg: TMessage); virtual id_First +
id_Ok;
```

Beantwortet das Anklicken des Schalters »OK« mit einem Aufruf von *CanClose* , um das Schließen des Fensters genehmigen zu lassen, und gibt das Dialogfenster anschließend frei.

Querverweis: *TDlgWindow.Cancel*

TDosStream

WObjects

Beschreibung

TDosStream ist ein spezialisierter *TStream*-Abkömmling, der ungepufferte DOS-Datei-Streams implementiert. Mit dem Konstruktor kann eine DOS-Datei durch Angeben ihres Namens und des Zugriffsmodus erzeugt und geöffnet werden: *stCreate, stOpenread, stOpenWrite* oder *stOpen*. Das einzige zusätzliche Feld von *TDosStream* ist *Handle* , das herkömmliche DOS-Datei-Handle zum Zugriff auf eine offene Datei. Die meisten Anwendungen benutzen *TBufStream,* den gepufferten Abkömmling von *TDosStream. TDosStream* überschreibt außer *TStream.Flush* alle abstrakten Methoden von *TStream.*

Felder

Handle

```
Handle: Word;
```

Handle ist das DOS-Datei-Handle, das verwendet wird, um auf einen offenen Datei-Stream zuzugreifen.

Methoden

Init

```
constructor Init(Filename: PChar; Mode: Word);
```

Erzeugt einen DOS-Datei-Stream mit dem angegebenen Dateinamen *FileName* und dem angegebenen Zugriffsmodus. Bei erfolgreicher Ausführung wird im Feld *Handle* das DOS-Datei-Handle gespeichert, andernfalls wird *Error* mit dem Argument *stInitError* für *Code* aufgerufen.

Das Argument *Mode* muß auf einen der Werte *stCreate,*

stOpenRead , *stOpenWrite* oder *stOpen* gesetzt werden. Diese Konstanten werden in Kapitel 6 »Außerhalb der Objekte« im Abschnitt »stXxxx« erläutert.

Done
```
destructor Done; virtual;
```
Schließt den DOS-Datei-Stream und gibt ihn frei.

Querverweis: *TDosStream.Init*

GetPos
```
function GetPos: LongInt; virtual;
```
Gibt den Wert der aktuellen Position des Streams zurück.

Querverweis: *TDosStream.Seek*

GetSize
```
function GetSize: LongInt; virtual;
```
Gibt die gesamte Größe des aufrufenden Streams in Byte zurück.

Read
```
procedure Read(var Buf; Count: Word); virtual;
```
Liest, beginnend an der aktuellen Position des aufrufenden Streams, *Count* Byte in den Puffer *Buf* ein.

Querverweis: *TDosStream.Write, stReadError*

Truncate
```
procedure Truncate; virtual;
```
Löscht alle Daten im aufrufenden Stream ab der aktuellen Position bis zum Ende.

Seek
```
procedure Seek(Pos: LongInt); virtual;
```
Setzt die aktuelle Position auf *Pos* Bytes ab dem Anfang des Streams.

Querverweis: *TDosStream.GetPos, TDosStream.Seek*

Write
```
procedure Write(var Buf; Count: Word); virtual;
```
Schreibt ab der aktuellen Position *Count* Byte aus dem Puffer *Buf* in den aufrufenden Stream.

Querverweis: *TDosStream.Read, stWriteError*

TEdit **WObjects**

Beschreibung *TEdit* ist ein Schnittstellenobjekt, das ein entsprechendes Editierfeld repräsentiert. *TEdit*-Objekte werden normalerweise nicht in Dialogfenstern (*TDialog* oder *TDlgWindow*) verwendet, sondern werden eingesetzt, wenn man ein selbständiges Editierfeld im Client-Bereich eines anderen Fensters darstellen will.

Es gibt zwei Stile von Editierfeldobjekten: einzeilig und mehrzeilig. Mehrzeilige Editierfelder ermöglichen vertikale

Bildlaufleisten und die Bearbeitung mehrerer Zeilen. Die meisten *TEdit*-Methoden verwalten den Text des Editierfeldes. *TEdit* schließt auch einige Methoden zur Beantworung von Befehlsbotschaften ein. Diese reagieren auf Menüauswahlen des übergeordneten Fensters des Editierfeldes wie zum Beispiel Ausschneiden, Kopieren, Einfügen, Ausfügen, Löschen und Rückgängig machen.

Zwei wichtige, vom Vorfahren *TStatic* geerbte Methoden, sind *GetText* und *SetText*.

Felder

IsMultiline

```
IsMultiline: Boolean;
```

IsMultiline ist im Fall eines Mehrzeileneditierfeldes *True*, ansonsten *False*.

Methoden

Init

```
constructor Init(AParent: PWindowsObject; AnId:
Integer; ATitle: PChar; X, Y, W, H: Integer; ATextLen:
Word; Multiline: Boolean);
```

Initialisiert ein Editierfeldobjekt für sein übergeordnetes Fenster (*AParent*) und füllt seine *Attr*-Felder mit der übergebenen Steuerelement-ID (*AnId*), dem Anfangstext (*ATitle*), der Position relativ zum Ursprung des Client-Bereichs des übergeordneten Fensters (*X,Y*), der Breite (*W*), der Höhe (*H*) und der Textpufferlänge (*ATextLen*).

Wenn *ATextLen* 0 ist, gibt es keine explizite Grenze für die Anzahl der Zeichen, die eingegeben werden können.

Wenn *Multiline True* ist, ist das Editierfeld ein mehrzeiliges Editierfeld mit horizontalen und vertikalen Bildlaufleisten. In diesem Fall schließt das Feld *Attr.Style* die Windows-Stilkonstanten *es_Multiline, es_AutoVScroll, es_AutoHScroll, es_Left, ws_VScroll* und *ws_HScroll* ein. Wenn *Multiline False* ist, hat das Editierfeld eine einzige Textzeile, eine Umrandung (*ws_Border*) und ist linksbündig (*es_Left*).

Load

```
constructor Load(var S: TStream);
```

Diese Methode initialisiert und lädt ein Editierfeld aus dem Stream *S*, indem sie zuerst *TStatic.Load* aufruft und dann das von *TEdit* hinzugefügte Feld (*IsMultiline*) liest.

Querverweis: *TStatic.Load*

CanUndo

```
function CanUndo: Boolean;
```

Gibt *True* zurück, wenn die letzte Bearbeitung rückgängig gemacht werden kann.

Querverweis: *TEdit.Undo*

ClearModify

```
procedure ClearModify;
```

Setzt das Änderungsflag für das Editierfeld zurück.

Querverweis: *TEdit.IsModified*

CMEditClear

```
procedure CMEditClear(var Msg: TMessage); virtual
cm_First + cm_EditClear;
```

Beantwortet die Auswahl einer Menüoption mit einer Menü-ID von *cm_EditClear* mit einem Aufruf der Methode *Clear*.

Querverweis: *TStatic.Clear*

CMEditCopy

```
procedure CMEditCopy(var Msg: TMessage); virtual
cm_First + cm_EditCopy;
```

Beantwortet die Auswahl einer Menüoption mit einer Menü-ID von *cm_EditCopy* mit einem Aufruf der Methode *Copy*.

Querverweis: *TEdit.Copy*

CMEditCut

```
procedure CMEditCut(var Msg: TMessage); virtual
cm_First + cm_EditCut;
```

Beantwortet die Auswahl einer Menüoption mit einer Menü-ID von *cm_EditCut* mit einem Aufruf der Methode *Cut*.

Querverweis: *TEdit.Cut*

CMEditDelete

```
procedure CMEditDelete(var Msg: TMessage); virtual
cm_First + cm_EditDelete;
```

Beantwortet die Auswahl einer Menüoption mit einer Menü-ID von *cm_EditDelete* mit einem Aufruf der Methode *DeleteSelection*.

Querverweis: *TEdit.DeleteSelection*

CMEditPaste

```
procedure CMEditPaste(var Msg: TMessage); virtual
cm_First + cm_EditPaste;
```

Beantwortet die Auswahl einer Menüoption mit einer Menü-ID von *cm_EditPaste* mit einem Aufruf der Methode *Paste*.

Querverweis: *TEdit.Paste*

CMEditUndo

```
procedure CMEditUndo(var Msg: TMessage); virtual
cm_First + cm_EditUndo;
```

Beantwortet die Auswahl einer Menüoption mit einer Menü-ID von *cm_EditUndo* mit einem Aufruf der Methode *Undo*.

Querverweis: *TEdit.Undo*

Copy

```
procedure Copy;
```

Kopiert den aktuell ausgewählten Text in die Zwischenablage.

Querverweis: *TEdit.CMEditCopy*

Cut

```
procedure Cut;
```

Schneidet den aktuell ausgewählten Text aus, das heißt er wird in die Zwischenablage kopiert und gelöscht.

Querverweis: *TEdit.CMEditCut*

DeleteLine

```
function DeleteLine(LineNumber: Integer): Boolean;
```

Löscht in einem Mehrzeileneditierfeld den Text in der durch *LineNumber* vorgegebenen Zeile. *DeleteLine* entfernt das Zeilenendezeichen nicht und beeinflußt auch keine anderen Zeilen. Diese Methode gibt bei erfolgreicher Ausführung *True* zurück.

DeleteSelection

```
function DeleteSelection: Boolean;
```

Löscht den aktuell ausgewählten Text und gibt *False* zurück, wenn kein Text ausgewählt ist.

Querverweis: *TEdit.CMEditDelete*

DeleteSubText

```
function DeleteSubText(StartPos, EndPos: Integer):
Boolean;
```

Löscht den Text, der zwischen der von *StartPos* und *EndPos* bezeichneten Start- und Endposition liegt. Das erste Zeichen steht an Position 0. Die Positionsnummern gehen kontinuierlich über alle Zeilen in einem Mehrzeileneditierfeld, Zeilenumbrüche zählen als zwei Zeichen. *DeleteSubText* gibt bei erfolgreicher Ausführung *True* zurück.

GetClassName

```
function GetClassName: PChar, virtual;
```

Gibt 'Edit' zurück, die Bezeichnung der Fensterklasse von *TEdit*.

GetLine

```
function GetLine(ATextString: PChar; StrSize,
LineNumber: Integer): Boolean;
```

Ruft den Text eines Mehrzeileneditierfeldes aus der mit *LineNumber* vorgegebenen Zeile ab und gibt ihn in *ATextString* zurück. *StrSize* zeigt an, wieviele Zeichen abgerufen werden sollen. *GetLine* gibt *False* zurück, wenn der Text nicht abgerufen werden kann oder wenn er zu lang ist.

Querverweis: *TStatic.GetText, TEdit.GetNumLines, TEdit.GetLineLength*

GetLineIndex

```
function GetLineIndex(LineNumber: Integer): Integer;
```

Gibt aus einem Mehrzeileneditierfeld die Anzahl der Zeichen zurück, die vor der von *LineNumber* bezeichneten Zeilennummer stehen. Zeilenumbrüche zählen als zwei Zeichen.

Wenn die angegebene Zeile nicht existiert, liefert *GetLineIndex* die Gesamtzahl der Zeichen im Editierfeld zurück.

GetLine-FromPos

```
function GetLineFromPos(CharPos: Integer): Integer;
```

Gibt aus einem Mehrzeileneditierfeld die Zeilennummer zurück, in der die von *CharPos* vorgegebene Zeichenposition enthalten ist. Die Zeichen werden, beginnend mit 0, fortlaufend von Zeile zu Zeile gezählt, Zeilenumbrüche zählen als zwei Zeichen.

GetLineLength

```
function GetLineLength(LineNumber: Integer): Integer;
```

Gibt aus einem Mehrzeileneditierfeld die Anzahl der Zeichen in der von *LineNumber* bezeichneten Zeile zurück. *GetLine-Length* sollte vor *GetLine* aufgerufen werden.

Querverweis: *TEdit.GetLine*

GetNumLines

```
function GetNumLines: Integer;
```

Gibt die Anzahl der Zeilen eines Mehrzeileneditierfeldes zurück. *GetNumLines* sollte vor *GetLine* aufgerufen werden.

Querverweis: *TEdit.GetLine*

GetSelection

```
procedure GetSelection(var StartPos, EndPos: Integer);
```

Ruft die Start- und Endpositionen des aktuell ausgewählten Textes ab und gibt diese in den Argumenten *StartPos* und *EndPos* zurück. Das erste Zeichen steht an Position 0. In einem Mehrzeileneditierfeld werden die Zeichen fortlaufend durch alle Zeilen gezählt, wobei Zeilenumbrüche als zwei Zeichen zählen. Durch Verwendung von *GetSelection* in Verbindung mit *GetSubText* können Sie den aktuell ausgewählten Text erhalten.

Querverweis: *TEdit.GetSubText*

GetSubText

```
procedure GetSubtext(ATextString: PChar; StartPos,
EndPos: Integer);
```

Liefert in *ATextString* den Text eines Editierfeldes vom Index *StartPos* bis zu *EndPos*. Das erste Zeichen steht an Position 0. In einem Mehrzeileneditierfeld werden die Zeichen fortlaufend durch alle Zeilen gezählt, wobei Zeilenumbrüche als zwei Zeichen gezählt werden.

Querverweis: *TEdit.GetSelection*

Insert

```
procedure Insert(ATextString: PChar);
```

Fügt den in *ATextString* übergebenen Text am aktuellen Einfügungspunkt in das Editierfeld ein und ersetzt gegebenenfalls den aktuell ausgewählten Text. *Insert* funktioniert ähnlich wie *Paste*, berührt jedoch nicht die Zwischenablage.

Querverweis: *TEdit.Paste*

IsModified `function IsModified: Boolean;`

Gibt *True* zurück, wenn der Benutzer den Text im Editierfeld geändert hat.

Querverweis: *TEdit.ClearModify*

Paste `procedure Paste;`

Fügt Text aus der Zwischenablage am aktuellen Einfügungspunkt in das Editierfeld ein.

Querverweis: *TEdit.CMEditPaste*

Scroll `procedure Scroll(HorizontalUnit, VerticalUnit: Integer);`

Rollt ein Mehrzeileneditierfeld horizontal und vertikal um die Anzahl von Zeichen, die in *HorizontalUnit* und *VerticalUnit* vorgegeben sind. Positive Werte bewirken ein Rollen nach rechts bzw. unten, negative Werte rollen nach links bzw. oben.

SetSelection `function SetSelection(StartPos, EndPos: Integer): Boolean;`

Bewirkt die Auswahl des Textes zwischen den Positionen, die von *StartPos* und *EndPos* vorgegeben sind, wobei das Zeichen an *EndPos* ausgeschlossen bleibt. Die Zeichen werden, beginnend mit 0, fortlaufend über alle Zeilen in einem Mehrzeileneditierfeld gezählt, wobei Zeilenumbrüche als zwei Zeichen gezählt werden.

SetupWindow `procedure SetupWindow; virtual;`

Begrenzt durch Aufruf von *TStatic.SetupWindow* die Anzahl der Zeichen, die in das Editierfeld eingegeben werden können. Wenn das Feld *TextLen* ungleich 0 ist, wird anschließend der Wert *TextLen* - 1 über eine *em_LimitText* -Botschaft an Windows gesendet.

Querverweis: *TWindow.SetupWindow, em_LimitText*-Botschaft

Store `procedure Store(var S: TStream);`

Diese Methode speichert das Editierfeld im Stream *S*, indem sie zuerst *TStatic.Store* aufruft und dann das von *TEdit* hinzugefügte Feld (*IsMultiline*) in den Stream schreibt.

Querverweis: *TStatic.Store*

Undo `procedure Undo;`

Macht die letzte Änderung rückgängig.

Querverweis: *TEdit.CanUndo, TEdit.CMEditUndo*

TEmsStream — WObjects

Beschreibung

TEmsStream ist ein Abkömmling von *TStream* zur Implementierung von Streams im EMS-Speicher. Die zusätzlichen Felder halten ein EMS-Handle, einen Seitenzähler, die Stream-Größe und die aktuelle Position bereit.

TEmsStream überschreibt die sechs abstrakten Methoden von *TStream* und stellt einen spezialisierten Konstruktor und Destruktor zur Verfügung.

Hinweis: Beim Debuggen eines Programms, das EMS-Streams verwendet, kann die IDE den von Ihrem Programm reservierten EMS-Speicher nicht wiederherstellen, wenn das Programm vorzeitig beendet wird oder wenn Sie vergessen, den Destruktor *Done* für einen EMS-Stream aufzurufen. Nur die Methode *Done* oder ein Neustart des Streams können die vom Stream belegten EMS-Seiten freigeben.

Felder

Handle

```
Handle: Word;
```

Das EMS-Handle für den Stream.

PageCount

```
PageCount: Word;
```

Die Anzahl der für den Stream reservierten Seiten mit 16 KByte pro Seite.

Position

```
Position: LongInt;
```

Dia aktuelle Position innerhalb des Streams. Die erste Position ist 0.

Size

```
Size: LongInt;
```

Die Größe des Streams in Byte.

Methoden

Init

```
constructor Init(MinSize, MaxSize: LongInt);
```

Diese Methode erzeugt einen EMS-Stream mit den vorgegebenen Minimal- und Maximalgrößen in Byte. Sie ruft *TStream.Init* auf und setzt dann *Handle, Size* und *PageCount.* Wenn die Initialisierung fehlschlägt, ruft sie *Error* mit dem Argument *stInitError* für *Code* auf

Querverweis: *TEmsStream.Done*

Done

```
destructor Done; virtual;
```

Gibt den EMS-Stream und die benutzten EMS-Speicherseiten frei.

Querverweis: *TEmsStream.Init*

GetPos
```
function GetPos: LongInt; virtual;
```
Gibt den Wert der aktuellen Position des aufrufenden Streams zurück.

Querverweis: *TEmsStream.Seek*

GetSize
```
function GetSize: LongInt; virtual;
```
Gibt die Gesamtgröße des aufrufenden Streams zurück.

Read
```
procedure Read(var Buf; Count: Word); virtual;
```
Liest, beginnend an der aktuellen Position des aufrufenden Streams, *Count* Bytes in den Puffer *Buf* ein.

Querverweis: *TEmsStream.Write, stReadError*

Seek
```
procedure Seek(Pos: LongInt); virtual;
```
Setzt die aktuelle Postion auf *Pos* Bytes vom Anfang des aufrufenden Streams.

Querverweis: *TEmsStream.GetPos, TEmsStream.GetSize*

Truncate
```
procedure Truncate; virtual;
```
Löscht alle Daten im aufrufenden Stream ab der aktuellen Position bis zum Ende. Die aktuelle Position wird anschließend auf das neue Ende des Streams gesetzt.

Querverweis: *TEmsStream.GetPos, TEmsStream.Seek*

Write
```
procedure Write(var Buf; Count: Word); virtual;
```
Schreibt, beginnend an der aktuellen Position, *Count* Bytes aus dem Puffer *Buf* in den aufrufenden Stream.

Querverweis: *TStream.Read, TEmsStream.GetPos, TEmsStream.-Seek*

TGroupBox — WObjects

Beschreibung

TGroupBox ist ein Schnittstellenobjekt, das eine entsprechende Gruppe repräsentiert. *TGroupBox*-Objekte werden normalerweise nicht in Dialogfenstern (*TDialog* oder *TDlgWindow*) verwendet, sondern dann, wenn man eine selbständige Gruppe im Client-Bereich eines anderen Fensters darstellen will.

Obwohl Gruppen keine aktive Rolle auf dem Bildschirm spielen, vereinigen sie visuell mehrere Elemente mit Auswahlmöglichkeiten (Auswahlfelder oder Schaltfelder). Im Hintergrund verwalten sie jedoch den Zustand der Elemente, die eine Auswahl anbieten. Zum Beispiel kann eine Gruppe auf das Auswählen eines Feldes durch den Benutzer mit der Rücknahme der Markierung aller anderen Felder reagieren.

Felder

NotifyParent

```
NotifyParent: Boolean;
```

Ein Flag, das anzeigt, ob das übergeordnete Fenster benachrichtigt werden soll, wenn sich der Zustand der Auswahlfelder der Gruppe geändert hat.

Methoden

Init

```
constructor Init(AParent: PWindowsObject; AnID:
Integer; AText: PChar; X, Y, W, H: Integer);
```

Diese Methode initialisiert das Gruppenobjekt für das übergebene übergeordnete Fenster (*AParent*), mit der Steuerelement-ID (*AnId*), dem zugeordneten Text (*AText*), der Position relativ zum Ursprung des Client-Bereichs des übergeordneten Fensters (*X,Y*), der Breite (*W*) und der Höhe (*H*). Sie ruft *TControl.Init* auf und fügt zum Feld *Attr.Style* des Gruppenfeldes den Windows-Stil *bs_GroupBox* hinzu und entfernt den Stil *ws_TabStop*. *NotifyParent* wird auf *True* gesetzt; per Voreinstellung wird das übergeordnete Fenster der Gruppe benachrichtigt, wenn sich der Zustand seiner Auswahlfelder ändert.

Querverweis: *TControl.Init*

InitResource

```
constructor InitResource(AParent: PWindowsObject;
ResourceID: Word);
```

Diese Methode nimmt eine Unterklassifizierung der Gruppe vor, indem sie ein ObjectWindows-Objekt initialisiert, das einer durch eine Dialogressource erzeugten Gruppe entsprechen soll.

Um die Gruppe vom Transfermechanismus auszuschließen, ruft sie *TControl.Initresource* und *DisableTransfer* auf, da sie keine zu transferierenden Daten enthält.

Querverweis: *TControl.InitResource, TWindowsObject.DisableTransfer*

Load

```
constructor Load(var S: TStream);
```

Initialisiert und lädt eine Gruppe aus dem Stream *S*, indem sie zuerst *TControl.Load* aufruft und dann das von *TGroupBox* hinzugefügte Feld (*NotifyParent*) liest.

Querverweis: *TWindow.Load*

GetClassName

```
function GetClassName: PChar; virtual;
```

Liefert 'Button' zurück, die Bezeichnung der Fensterklasse von *TGroupBox*.

SelectionChanged

```
procedure SelectionChanged(ControlId: Integer);
virtual;
```

Wenn *NotifyParent True* ist, benachrichtigt diese Methode das übergeordnete Fenster der Gruppe durch eine Botschaft auf Basis der ID untergeordneter Fenster, daß sich eine seiner Auswahlen geändert hat. Diese Methode kann überschrieben werden, damit die Gruppe auf seine Auswahlen reagieren kann.

Store

```
procedure Store(var S: TStream);
```

Speichert die Gruppe durch Aufruf von *TControl.Store* und schreibt dann das von *TGroupBox* hinzugefügte Feld (*NotifyParent*) in den Stream.

Querverweis*: TTWindow.Store*

TListBox — WObjects

Beschreibung

TListBox ist ein Schnittstellenobjekt, das eine entsprechende Liste repräsentiert. *TListBox*-Elemente werden normalerweise nicht in Dialogfenstern (*TDialog* oder *TDlgWindow*) verwendet, aber sie werden benutzt, wenn man eine selbständige Liste im Client-Bereich eines anderen Fensters darstellen will. Die Methoden von *TListBox* dienen auch den Instanzen seines Nachkommens, *TComboBox*.

Methoden

Init

```
constructor Init(AParent: PWindowsObject; AnId:
Integer; X, Y, W, H: Integer);
```

Diese Methode initialisiert ein Listenfensterobjekt für das übergebene übergeordnete Fenster (*AParent*) mit der Steuerelement-ID (*AnId*), der Position relativ zum Ursprung des Client-Bereichs des übergeordneten Fensters (*X,Y*), der Breite (*W*) und der Höhe (*H*).

Sie ruft *TControl.Init* auf und fügt zum Feld *Attr.Style* des Listenobjekts die Windows-Stilkonstante *lbs_Standard* hinzu. Diese hat für die Liste folgende Bedeutung:

- Eine Umrandung (*ws_Border*)
- Eine vertikale Bildlaufleiste (*ws_VScroll*)
- Automatisches alphabetisches Sortieren der Listenelemente (*lbs_Sort*)
- Benachrichtigung des übergeordneten Fensters nach einer Auswahl (*lbs_Notify*)

Diese Stile können in einem Nachkommen der Klasse oder im Konstruktor *Init* des der Liste übergeordneten Fensterobjekts überschrieben werden.

AddString
```
function AddString(AString: PChar): Integer;
```
Fügt *AString* als ein Listenelement zum Listenobjekt hinzu und gibt den Positionsindex des Elements (beginnend bei Null) zurück. Im Fall eines Fehlers wird ein negativer Wert zurückgegeben. Die Listenelemente werden automatisch sortiert, es sei denn, der Stil *lbs_Sort* wurde vor der Erzeugung aus dem Feld *Attr.Style* des Listenobjekts gelöscht.

ClearList
```
procedure ClearList;
```
Löscht alle Listenelemente aus der Liste.

DeleteString
```
function DeleteString(Index: Integer): Integer;
```
Löscht das Listenelement an der Position *Index* (beginnend bei Null). *DeleteString* gibt die Anzahl der verbleibenden Listenelemente zurück, im Fall eines Fehlers wird ein negativer Wert zurückgegeben.

GetClassName
```
function GetClassName: PChar; virtual;
```
Gibt 'ListBox', die Bezeichnung der Fensterklasse von *TListBox* zurück.

GetCount
```
function GetCount: Integer;
```
Gibt die Anzahl der Listenelemente in der Liste zurück, im Fall eines Fehlers wird ein negativer Wert zurückgeliefert.

GetSelIndex
```
function GetSelIndex: Integer;
```
Gibt den Positionsindex (beginnend bei Null) des aktuell ausgewählten Listenelements bzw. einen negativen Wert, wenn kein Element ausgewählt ist zurück.

GetSelString
```
function GetSelString(AString: PChar; MaxChars:
Integer): Integer;
```
Liefert in *AString* das aktuell ausgewählte Listenelement zurück, falls dieses nicht länger als *MaxChars* ist. *GetSelString* gibt die Stringlänge, bzw. im Fall eines Fehlers einen negativen Wert zurück.

GetString
```
function GetString(AString: PChar; Index: Integer):
Integer;
```
Liefert in *AString* das Listenelement an der Position *Index* (beginnend bei Null) und gibt die Stringlänge zurück. Im Fall eines Fehlers wird ein negativer Wert zurückgeliefert.

GetStringLen
```
function GetStringLen(Index: Integer): Integer;
```
Gibt die Stringlänge des Listenelements an der Position *Index* zurück. Im Fall eines Fehlers wird ein negativer Wert zurückgeliefert.

InsertString

```
function InsertString(AString: PChar; Index: Integer):
Integer;
```

Fügt *AString* als ein Listenelement an der Position *Index* in die Liste ein und gibt die aktuelle Position des Elements (beginnend bei 0) zurück. Im Fall eines Fehlers wird ein negativer Wert zurückgeliefert. Die Listenelemente werden nicht neu sortiert. Wenn *Index* -1 ist, wird der String an das Ende der Liste hinzugefügt.

SetSelIndex

```
function SetSelIndex(Index: Integer): Integer; virtual;
```

Bewirkt die Auswahl des Listenelements an der Position *Index* (beginnend bei 0). Wenn *Index* -1 ist, wird aus der Liste jede Auswahl gelöscht. *SetSelIndex* gibt im Fall eines Fehlers eine negative Zahl zurück.

SetSelString

```
function SetSelString(AString: PChar; Index: Integer):
Integer;
```

Bewirkt die Auswahl des ersten Listenelements, das mit dem in *AString* übergebenen Text übereinstimmt ab der Position *AIndex* (beginnend bei Null). Ist *Index* auf -1 gesetzt, wird die gesamte Liste von Anfang an durchsucht. *SetSelString* gibt den Positionsindex des neu ausgewählten Elements zurück, im Fall eines Fehlers wird ein negativer Wert zurückgeliefert.

Transfer

```
function Transfer(DataPtr: Pointer; TransferFlag:
Word): Word; virtual;
```

Transferiert die Liste der Einträge und ausgewählten Elemente zu oder von dem Transfer-Record, auf den *DataPtr* zeigt. Ist *TransferFlag tf_GetData*, werden die Daten des Listenfensters in den Record transferiert. Ist *TransferFlag tf_SetData*, wird die Liste mit dessen Daten geladen. *Transfer* gibt die Anzahl der transferierten Bytes zurück. Ist *TransferFlag tf_SizeData*, gibt *Transfer* die Größe der Transferdaten zurück. Das Wesen des Records variiert etwas, abhängig davon, ob das Fenster zuläßt, daß mehrere Elemente ausgewählt sind. Das erste transferierte Element ist immer ein Zeiger auf eine Kollektion von Strings, die die Einträge der Liste darstellen. Bei Listen mit Einzelauswahl folgt diesem ein Integerindex auf das ausgewählte Element. Bei Listen mit Mehrfachauswahl folgt auf die Kollektion ein Zeiger auf einen *TMultiSelRec*-Record, der ein Array von Integern enthält, von denen jeder ein ausgewähltes Element kennzeichnet. Typische Transfer-Records sehen wie folgt aus:

```
type
  TListBoxXferRec =     record              {Einzelauswahl}
    Strings: PStrCollection;
    Selection: Integer;
  end ;
```

```
TMultiListXferRec =     record          { Mehrfachauswahl}
  Strings: PStrCollection;
  Selections: PMultiSelRec;
end ;
```

PMultiSelRec ist ein Record, der in der Unit *WObjects* definiert ist. Der Speicher für den Record muß unter Verwendung der Funktion *AllocMultiSel* zugewiesen werden und nach dem Transfer mit *FreeMultiSel* freigegeben werden. Ein **nil**-Zeiger zeigt an, daß keine Einträge ausgewählt sind.

TMDIClient — WObjects

Beschreibung

MDI-Client-Fenster werden von *TMDIClient* repräsentiert und sind Steuerelemente, die die untergeordneten MDI-Fenster einer MDI-Anwendung verwalten. Die Methoden von *TMDIClient* sind mit der Verwaltung untergeordneter MDI-Fenster beschäftigt.

Felder

ClientAttr

```
ClientAttr: TClientCreateStruct;
```

ClientAttr enthält einen Record mit den Attributen des MDI-Client-Fensters. *TClientCreateStruct* ist definiert als:

```
type
  PClientCreateStruct = ^TClientCreateStruct;
  TClientCreateStruct =     record
    hWindowMenu: THandle;
    idFirstChild: Word;
  end ;
```

Methoden

Init

```
constructor Init(AParent: PMDIWindow);
```

Initialisiert das MDI-Client-Fensterobjekt mit *AParent* als dem übergeordneten Fenster. *Init* setzt die Felder von *ClientAttr*, ruft *TControl.Init* auf und fügt den Windows-Stil *ws_ClipChildren* zum Feld *Attr.Style* des Fensterobjekts hinzu. Außerdem löscht *Init* das Client-Fenster aus der Liste der untergeordneten Fenster seines übergeordneten Fensters, so daß es nicht wie andere untergeordnete Fenster – wie etwa Listen und Schalter – behandelt wird.

Querverweis: *TControl.Init*

Load

```
constructor Load(var S: TStream);
```

Diese Methode initialisiert und lädt ein MDI-Client-Fenster aus dem Stream *S*, indem sie zuerst *TControl.Load* aufruft und dann das von *TMDIClient* hinzugefügte Feld (*ClientAttr*) liest.

Querverweis: *TWindow.Load*

ArrangeIcons
```
procedure ArrangeIcons; virtual;
```
Richtet die minimierten untergeordneten MDI-Fenster am unteren Rand des MDI-Client-Fensters in einer ordentlichen Reihe aus.

CascadeChildren
```
procedure CascadeChildren; virtual;
```
Ordnet alle nicht minimierten untergeordneten MDI-Fenster innerhalb des MDI-Client-Fensters so an und stellt deren Grösse so ein, daß sie sich zum großen Teil überlappen, die Titelleisten aber alle sichtbar sind.

GetClassName
```
function GetClassName: PChar; virtual;
```
Gibt 'MDIClient', die Fensterklassenbezeichnung von *TMDIClient*, zurück.

Store
```
procedure Store(var S: TStream);
```
Speichert das MDI-Client-Fenster im Stream *S*, indem sie zuerst *TControl.Store* aufruft und dann das von *TMDIClient* hinzugefügte Feld (*ClientAttr*) in den Stream schreibt.

Querverweis: *TWindow.Store*

TileChildren
```
procedure TileChildren; virtual;
```
Ordnet alle nicht minimierten untergeordneten MDI-Fenster innerhalb des MDI-Client-Fensters so an und stellt deren Grösse so ein, daß sie den gesamten zur Verfügung stehenden Platz einnehmen, ohne sich zu überlappen.

TMDIWindow **WObjects**

Beschreibung MDI-Rahmenfenster werden von *TMDIWindow* repräsentiert und sind überlappte Fenster, die als Hauptfenster für MDI-Anwendungen dienen. Ein Hauptmerkmal von *TMDIWindow*-Objekten ist, daß sie ein *TMDIClient*-Objekt besitzen und es im Feld *ClientWnd* speichern. Ein anderes Merkmal ist ein Menü, das Optionen zur Manipulation der untergeordneten MDI-Fenster der Anwendung bietet. Dieses Fenster wird automatisch modifiziert, um alle dargestellten untergeordneten MDI-Fenster widerzuspiegeln.

Felder

ChildMenuPos
```
ChildMenuPos: Integer;
```
ChildMenuPos bezeichnet einen Index, der die Position des Verwaltungsmenüs der untergeordneten Fenster kennzeichnet. Der Index zählt nur Hauptmenüoptionen, wobei die oberste linke Option an Position Null steht.

ClientWnd

```
ClientWnd: PMDIClient;
```

ClientWnd zeigt auf das MDI-Client-Fenster des MDI-Rahmenfensters, einer Objektinstanz von *TMDIClient*.

Methoden

Init

```
constructor Init(ATitle: PChar; AMenu: HMenu);
```

Initialisiert ein MDI-Rahmenfensterobjekt unter Verwendung der Titelzeile *ATitle* und des Menüs *AMenu*. Ein MDI-Rahmenfenster hat kein übergeordnetes Fenster, es muß das Hauptfenster der Anwendung sein und über ein Menü verfügen. *Init* setzt als Voreinstellung *ChildMenuPos* auf 0 und kennzeichnet damit das oberste linke Menü als das Menü zur Verwaltung der untergeordneten Fenster. Um *ChildMenuPos* zu modifizieren, überschreiben Sie *TMDIWindow.Init* mit den von Ihnen definierten Nachkommen der Typen. Beispiel:

```
constructor  MyMDIWindow.Init(ATitle: PChar; AMenu:
HMenu);
begin
  TMDIWindow.Init(ATitle, AMenu);
  ChildMenuPos := 3;
end ;
```

Querverweis: *TMDIWindow.InitClientWindow*

Load

```
constructor Load(var S: TStream);
```

Diese Methode initialisiert und lädt ein MDI-Rahmenfenster aus dem Stream *S*, indem sie zuerst *TWindow.Load* aufruft und dann die von *TMDIWindow* hinzugefügten Felder holt und liest (*ClientWnd* und *ChildMenuPos*).

Querverweis: *TWindow.Load*

Done

```
destructor Done; virtual;
```

Diese Methode gibt das in *ClientWnd* gespeicherte MDI-Client-Fensterobjekt frei, bevor sie *TWindow.Done* aufruft, um das MDI-Rahmenfensterobjekt freizugeben.

Querverweis: *TWindow.Done*

ArrangeIcons

```
procedure ArrangeIcons;
```

Richtet die minimierten untergeordneten MDI-Fenster in einer ordentlichen Reihe am unteren Rand des MDI-Client-Fensters aus. Ruft *ClientWnd^.ArrangeIcons* auf.

Querverweis: *TMDIClient.ArrangeIcons*

Cascade-Children

```
procedure CascadeChildren; virtual;
```

Ordnet alle nicht minimierten untergeordneten MDI-Fenster innerhalb des MDI-Client-Fensters so an, daß sie sich über-

lappen und die Titelleiste eines jeden dargestellt wird. Ruft *ClientWnd^.CascadeChildren* auf.

Querverweis: *TMDIClient.CascadeChildren*

CloseChildren

```
procedure CloseChildren; virtual;
```

Entfernt alle erzeugten untergeordneten MDI-Fenster, für die *CanClose True* zurückgibt.

CMArrange-Icons

```
procedure CMArrangeIcons(var Msg: TMessage); virtual
cm_First + cm_ArrangeIcons;
```

Beantwortet die Auswahl eines Menüs mit der ID *cm_Arrange-Icons* mit einem Aufruf von *ArrangeIcons*.

Querverweis: *TMDIWindow.ArrangeIcons*

CMCascade-Children

```
procedure CMCascadeChildren(var Msg: TMessage); virtual
cm_First + cm_CascadeChildren;
```

Beantwortet die Auswahl eines Menüs mit der ID *cm_Cascade-Children* mit einem Aufruf von *CascadeChildren*.

Querverweis: *TMDIWindow.CascadeChildren*

CMClose-Children

```
procedure CMCloseChildren(var Msg: TMessage); virtual
cm_First + cm_CloseChildren;
```

Beantwortet die Auswahl eines Menüs mit der ID *cm_Close-Children* mit einem Aufruf von *CloseChildren*.

Querverweis: *TMDIWindow.CloseChildren*

CMCreate-Child

```
procedure CMCreateChild(var Msg: TMessage); virtual
cm_First + cm_CreateChild;
```

Beantwortet die Auswahl eines Menüs mit der ID *cm_Create-Child* mit einem Aufruf *CreateChild*, um ein neues untergeordnetes Fenster zu erzeugen.

Querverweis: *TMDIWindow.CreateChild*

CMTile-Children

```
procedure CMTileChildren(var Msg: TMessage); virtual
cm_First + cm_TileChildren;
```

Beantwortet die Auswahl eines Menüs mit der ID *cm_Tile-Children* mit einem Aufruf von *TileChildren*.

Querverweis: *TMDIWindow.TileChildren*

CreateChild

```
function CreateChild: PWindowsObject; virtual;
```

Initialisiert und erzeugt durch Aufruf von *InitChild* und *Make-Window* ein neues untergeordnetes MDI-Fenster. Sie müssen *CreateChild* nicht überschreiben, um Nachkommen des untergeordneten MDI-Fenstertyps anzupassen, wie dies für *TMDI-*

Window.InitChild gilt. *CreateChild* gibt einen Zeiger auf das neue untergeordnete MDI-Fenster zurück.

Querverweis: *TMDIWindow.InitChild, TApplication.MakeWindow*

DefWndProc

```
procedure DefWndProc(var Msg: TMessage); virtual;
```

Überschreibt die Standardbotschaftsverarbeitung von *TWindow* durch Aufruf der Windows-Funktion *DefFrameProc* anstelle von *DefWindowProc*.

Querverweis: *TWindow.DefWndProc*

GetClassName

```
function GetClassName; PChar; virtual;
```

Gibt 'TurboMDIWindow' zurück, die Bezeichnung der Fensterklasse von *TMDIWindow*.

GetClient

```
function GetClient: PMDIClient; virtual;
```

Gibt einen Zeiger auf das MDI-Client-Fenster zurück, das in *ClientWnd* gespeichert ist.

GetWindowClass

```
procedure GetWindowClass(var AWndClass: TWndClass);
virtual;
```

Verändert den Standardfensterklassen-Record und gibt ihn in *AWndClass* zurück. *GetWindowClass* setzt das Stil-Feld zu 0, um die von *TWindow.GetWindowClass* gesetzten Stile zu löschen.

Querverweis: *TWindow.GetWindowClass*

InitChild

```
function InitChild: PWindowsObject; virtual;
```

Initialisiert ein untergeordnetes MDI-Fensterobjekt (*TWindow*) mit einer Überschrift 'MDI Child' und gibt einen Zeiger darauf zurück. Wenn Sie einen untergeordneten MDI-Fenstertyp definieren, der von *TWindow* abstammt, sollten Sie *TMDIWindow.InitChild* überschreiben, um ein Fenster mit Ihren neuen untergeordneten MDI-Fenstertypen zu erzeugen. Beispiel:

```
function  MyMDIWindow.InitChild: PWindowsObject;
begin
  InitChild := New(PmyMDIChild, Init(@Self,
               'Untitled Window'));
end ;
```

Querverweis: *TMDIWindow.CreateChild*

InitClientWindow

```
procedure InitClientWindow; virtual;
```

Initialisiert das MDI-Client-Fenster als ein *TMDIClient*-Objekt und speichert es in *ClientWnd*.

SetupWindow `procedure SetupWindow; virtual;`

Initialisiert durch Aufruf von *InitClientWindow* das dem MDI-Client-Fensterobjekt (*ClientWnd*) entsprechende Fensterelement und erzeugt dieses durch Aufruf von *MakeWindow*. Wenn Sie *TMDIWindow.SetupWindow* in einem abgeleiteten Typ überschreiben, rufen Sie dort *TMDIWindow.SetupWindow* explizit auf.

Querverweis: *TMDIWindow.InitClientWindow, TApplication.-MakeWindow*

Store `procedure Store(var S: TStream);`

Diese Methode speichert das MDI-Rahmenfenster im Stream *S*, indem sie zuerst *TWindow.Store* aufruft und dann die von *TMDIWindow* hinzugefügten Felder (*ClientWnd* und *ChildMenuPos*) in den Stream schreibt.

Querverweis*: TWindow.Store*

TileChildren `procedure TileChildren; virtual;`

Ordnet alle nicht minimierten untergeordneten MDI-Fenster innerhalb des MDI-Client-Fensters so an und bemißt ihre Größe so, daß sie den gesamten verfügbaren Platz einnehmen, ohne sich zu überlappen. Ruft *ClientWnd^.TileChildren* auf.

Querverweis: *TMDIClient.TileChildren*

TObject

WObjects

Beschreibung *TObject* ist der Ausgangspunkt der Objekthierarchie von ObjectWindows. Als Basisobjekt hat es keine Vorfahren, aber viele Nachkommen. Alle ObjectWindows-Standardobjekte stammen letztendlich von *TObject* ab. Jedes Objekt, das die ObjectWindows-Stream-Einrichtungen nutzt, muß seine Abstammung auf *TObject* zurückführen.

Methoden

Init `constructor init;`

Diese Methode reserviert auf dem Heap Speicher für das Objekt. Sie wird von allen Konstruktoren abgeleiteter Objekte aufgerufen.

Free `procedure Free;`

Gibt das Objekt frei und ruft den Destruktor *Done* auf.

Done `destructor Done; virtual;`

Führt Speicherfreigabe für dynamische Objekte aus.

TRadioButton **WObjects**

Beschreibung

TRadioButton ist ein Schnittstellenobjekt, das ein entsprechendes Schaltfeld repräsentiert. *TRadioButton*-Objekte werden normalerweise nicht in Dialogfenstern (*TDialog* oder *TDlgWindow*) verwendet, aber sie werden eingesetzt, wenn man ein selbständiges Schaltfeld im Client-Bereich eines anderen Fensters darstellen will.

Schaltfelder haben zwei Zustände: »markiert« und »nicht markiert«. *TRadioButton* erbt seine Methoden zur Verwaltung dieser Zustände von seinem Vorfahren *TCheckBox*. Optional kann ein Auswahlfeld Teil einer Gruppe (*TGroupBox*) sein, die ihre Steuerelemente visuell und funktionell gruppiert.

Methoden

Init

```
constructor Init(AParent: PWindowsObject; AnID:
Integer; ATitle: PChar; X, Y, W, H: Integer; AGroup:
PGroupBox);
```

Initialisiert ein Schaltfeldobjekt für das übergebene übergeordnete Fenster (*AParent*) mit der Steuerelement-ID (*AnId*), dem zugehörigen Text (*ATitle*), der Position relativ zum Ursprung des Client-Bereichs des übergeordneten Fensters (*X,Y*), der Breite (*W*), der Höhe (*H*) und der zugehörigen Gruppe (*AGroup*).

TRadioButton.Init setzt das Feld *Attr.Style* des Auswahlfeldes auf *ws_child* **or** *ws_Visible* **or** *bs_RadioButton*.

TScrollBar **WObjects**

Beschreibung

TScrollBar-Objekte stehen für selbständige Bildlaufleisten, aber nicht für die Bildlaufleisten, die mit Fenstern gekoppelt sind. Bildlaufleisten können vertikal (nach oben und unten rollend) oder horizontal (nach rechts und links rollend) sein. Die meisten Methoden von *TScrollBar* sind für die Verwaltung der Position und des Bewegungsbereichs des Bildlaufsymbols zuständig.

Ein spezielles Merkmal des Typs *TScrollBar* sind die Methoden, die die Windows-Bildlaufleistenbotschaften *wm_HScroll* und *wm_VScroll* beantworten. Diese Methoden, wie etwa *SBLineUp* und *SBPageDown*, sind als benachrichtigungsbasierte Methoden definiert und dienen der korrekten Positionierung des Bildlaufsymbols.

Hinweis: *TScrollBar*-Objekte sollten nie in Fenstern plaziert werden, die in ihren Attributen einen der Stile *ws_HScroll* oder *ws_VScroll* haben.

Felder

IsHorizontal
```
IsHorizontal: Boolean;
```
Dieses Feld ist *True*, wenn die Bildlaufleiste horizontal ist, und *False*, wenn sie vertikal ist.

LineMagnitude
```
LineMagnitude: Integer;
```
LineMagnitude ist die Anzahl der Bereichseinheiten, um die die Bildlaufleiste gerollt werden soll, wenn der Benutzer durch Anklicken der Pfeilsymbole der Bildlaufleiste eine kleine Bewegung wünscht. *Init* setzt per Voreinstellung *LineMagnitude* auf 1. *TScrollBar.SetupWindow* setzt den Rollbereich per Voreinstellung auf 0 bis 100.

Querverweis: *TScrollBar.SetupWindow*

PageMagnitude
```
PageMagnitude: Integer;
```
PageMagnitude ist die Anzahl der Bereichseinheiten, um die die Bildlaufleiste rollen soll, wenn der Benutzer eine weite Bewegung durch Anklicken im Rollbereich der Bildlaufleiste anfordert. *TScrollBar.Init* setzt *PageMagnitude* per Voreinstellung auf 10. (Der Rollbereich wird per Voreinstellung auf 0 bis 100 gesetzt.)

Methoden

Init
```
constructor Init(AParent: PWindowsObject; AnID:
Integer; X, Y, W, H: Integer; IsHScrollBar: Boolean);
```
Initialisiert ein *TScrollBar*-Objekt für das angegebene übergeordnete Fenster (*AParent*) mit *AnID* als Steuerelement-ID, einer Position von (*X,Y*), einer Breite von W und einer Höhe von *H*. Die Bildlaufleiste ist horizontal (Stil *sbs_Horz*), wenn *IsHScrollBar True* ist, andernfalls ist sie vertikal (Stil *sbs_Vert*). Wenn die übergebene Höhe für eine horizontale Bildlaufleiste oder die übergebene Breite für eine vertikale Bildlaufleiste 0 ist, wird ein Standardwert benutzt. *LineMagnitude* wird mit 1, *PageMagnitude* mit 10 initialisiert.

Load
```
constructor Load(var S: TStream);
```
Diese Methode initialisiert und lädt eine Bildlaufleiste aus dem Stream *S*, indem sie zuerst *TControl.Load* aufruft und dann die von *TScrollBar* hinzugefügten Felder liest (*IsHorizontal, LineMagnitude* und *PageMagnitude*).

Querverweis: *TWindow.Load*

DeltaPos
```
function DeltaPos(Delta: Integer): Integer;
```
Ändert mit *SetPosition* die Position des Bildlaufsymbols um den in *Delta* übergebenen Wert. Ein negativer Wert bewegt das

Symbol nach oben oder nach links. Die neue Position des Symbols wird zurückgegeben.

GetClassName
```
function GetClassName: PChar; virtual;
```
Gibt 'Scrollbar' zurück, die Bezeichnung der Fensterklasse von *TScrollBar*.

GetPosition
```
function GetPosition: Integer;
```
Gibt die aktuelle Position des Bildlaufsymbols zurück.

Querverweis: *TScrollBar.SetPosition*

GetRange
```
procedure GetRange(var LoVal, HiVal: Integer);
```
Liefert den zulässigen Bereich der Positionen der Bildlaufleiste in *LoVal* und *HiVal* zurück.

Querverweis: *TScrollBar.SetRange*

SBBottom
```
procedure SBBottom(var Msg: TMessage); virtual nf_First
+ sb_Bottom;
```
Setzt als Reaktion auf die Anforderung des Benutzers die Position des Bildlaufsymbols mit *SetPosition* auf den höchsten zulässigen Wert.

Diese Methode wird als Antwort auf eine bildlaufbasierte Botschaft mit dem Code *sb_Bottom* aufgerufen.

SBLineDown
```
procedure SBLineDown(var Msg: TMessage); virtual
nf_First + sb_LineDown;
```
Bewegt mit *SetPosition* das Bildlaufsymbol um *LineMagnitude* nach unten oder rechts.

SBLineDown wird als Reaktion auf eine bildlaufbasierte Botschaft mit dem Code *sb_LineDown* aufgerufen.

SBLineUp
```
procedure SBLineUp(var Msg: TMessage); virtual nf_First
+ sb_LineUp;
```
Bewegt mit *SetPosition* das Bildlaufsymbol um *LineMagnitude* nach oben oder nach links.

SBLineUp wird als Reaktion auf eine bildlaufbasierte Botschaft mit dem Code *sb_LineUp* aufgerufen. Diese Botschaft wird beispielsweise gesendet, wenn der Benutzer das Pfeilsymbol »nach oben« anklickt.

SBPageDown
```
procedure SBPageDown(var Msg: TMessage); virtual
nf_First + sb_PageDown;
```
Bewegt mit *SetPosition* das Bildlaufsymbol um *PageMagnitude* nach unten oder nach rechts.

SBPageDown wird automatisch als Reaktion auf eine bildlaufbasierte Botschaft mit dem Code *sb_PageDown* aufgerufen.

SBPageUp

```
procedure SBPageUp(var Msg: TMessage); virtual nf_First
+ sb_PageUp;
```

Bewegt mit *SetPosition* das Bildlaufsymbol um *PageMagnitude* nach oben oder nach links.

SBPageUp wird automatisch als Reaktion auf eine bildlaufbasierte Botschaft mit dem Code *sb_PageUp* aufgerufen.

SBThumb-Position

```
procedure SBThumbPosition(var Msg: TMessage); virtual
nf_First + sb_ThumbPosition;
```

Ändert mit *SetPosition* die Position des Bildlaufsymbols in die vom Benutzer gewählte und über eine bildlaufbasierte Botschaft mit dem Code *sb_ThumbPosition* übermittelte Position.

SBThumb-Track

```
procedure SBThumbTrack(var Msg: TMessage) ; virtual
nf_First + sb_ThumbTrack;
```

Ändert mit *SetPosition* die Position des Bildlaufsymbols, wenn es der Benutzer mit der Maus zieht.

Diese Methode wird als Reaktion auf eine bildlaufbasierte Botschaft mit dem Code *sb_ThumbTrack* aufgerufen.

SBTop

```
procedure SBTop(var Msg: TMessage); virtual nf_First +
sb_Top;
```

Setzt als Reaktion auf die Anforderung des Benutzers mit *SetPosition* die Position des Bildlaufsymbols auf den niedrigsten zulässigen Wert.

Diese Methode wird als Reaktion auf eine bildlaufbasierte Botschaft mit dem Code *sb_Top* aufgerufen.

SetPosition

```
procedure SetPosition(ThumbPos: Integer);
```

Setzt die Position des Bildlaufsymbols auf *ThumbPos*. Wenn *ThumbPos* den zulässigen Bereich nach oben bzw. unten überschreitet, wird die Postition des Bildlaufsymbols auf den höchsten bzw. niedrigsten zulässigen Wert gesetzt.

Querverweis: *TScrollBar.GetPosition*

SetRange

```
procedure SetRange(LoVal, HiVal: Integer);
```

Setzt den zulässigen Bereich der Positionen des Bildlaufsymbols auf *LoVal* bis *HiVal* fest.

Querverweis*: TScrollBar.GetRange*

SetupWindow

```
procedure SetupWindow; virtual;
```

Initialisiert den Bereich der Bildlaufleiste auf 0 bis 100. Um diesen Bereich zu überschreiben, rufen Sie *TScrollbar.SetRange* auf.

Querverweis: *TScrollBar.SetRange*

Store

```
procedure Store(var S: TStream);
```

Diese Methode speichert die Bildlaufleiste auf dem Stream *S*, indem sie zuerst *TControl.Store* aufruft und dann die von *TScrollBar* hinzugefügten Felder (*IsHorizontal*, *LineMagnitude* und *PageMagnitude*) in den Stream schreibt.

Querverweis: *TWindow.Store*

Transfer

```
function Transfer(DataPtr: Pointer; TransferFlag:
Word): Word; virtual;
```

Transferiert den Bereich der Bildlaufleiste und die aktuelle Position des Bildlaufsymbols zu oder von der Speicherstelle, auf die *DataPtr* zeigt. Wenn *TransferFlag tf_GetData* ist, werden diese Daten dort in einem Record gespeichert. Wenn *TransferFlag tf_SetData* ist, werden sie aus diesem Record abgerufen und verwendet, um den Bereich und die Position der Bildlaufleiste festzulegen. *Transfer* gibt die Anzahl der Bytes zurück, die an der Speicherstelle abgelegt oder von dort abgerufen wurden. Der spezielle Record ist folgendermaßen definiert:

```
ScrollBarTransferRec =     record
  LowValue: Integer;
  HighValue: Integer;
  Position: Integer;
end ;
```

TScroller

WObjects

Beschreibung

TScroller-Objekte gibt es im Feld *Scroller* von *TWindow* und in davon abgeleiteten Objekten. Das *TScroller*-Objekt bietet einem Fenster einen automatischen Bildlaufmechanismus, der in Verbindung mit oder ohne horizontale und vertikale Bildlaufleisten arbeitet. Es unterstützt ein als »automatisches Rollen« bezeichnetes Verfahren, das keine Bildlaufleisten erfordert.

Normalerweise initialisieren und manipulieren Sie *TScroller*-Objekte von den Methoden der Fensterobjekte aus, zu denen sie gehören.

Felder

Window

```
Window: PWindow;
```

Window zeigt auf das Fenster, das *TScroller* besitzt.

XUnit

```
XUnit: Integer,
```

XUnit ist die kleinste Anzahl von Geräteeinheiten (Pixeln), um die das Fenster horizontal gerollt werden kann. Die Werte von *XUnit* sind im Konstruktor *Init* vorgegeben, können aber später durch den Aufruf von Methoden modifiziert werden.

YUnit `YUnit: Integer;`

YUnit ist die kleinste Anzahl von Geräteeinheiten (Pixeln), um die das Fenster vertikal gerollt werden kann. Die Werte von *YUnit* sind im Konstruktor *Init* vorgegeben, können aber später durch den Aufruf von Methoden modifiziert werden.

XPos `XPos: LongInt;`

XPos ist die aktuelle horizontale Position, angegeben in *XUnit*-Einheiten.

YPos `Ypos: LongInt;`

YPos ist die aktuelle vertikale Position, angegeben in *YUnit*-Einheiten.

XRange `XRange: LongInt;`

XRange ist die Gesamtzahl von horizontalen *XUnit*-Einheiten, um die das Fenster gerollt werden kann. *XRange*-Werte sind im Konstruktor *Init* vorgegeben, können aber später modifiziert werden.

YRange `YRange: LongInt;`

YRange ist die Gesamtzahl von vertikalen *YUnit*-Einheiten, um die das Fenster gerollt werden kann. *YRange*-Werte sind im Konstruktor *Init* vorgegeben, können aber später modifiziert werden.

XLine `XLine: Integer;`

XLine ist die Anzahl der *XUnit*-Einheiten, um die als Reaktion auf ein Anklicken des Pfeilsymbols der Bildlaufleiste horizontal gerollt werden soll. Der voreingestellte Wert ist 1.

YLine `YLine: Integer;`

YLine ist die Anzahl der *YUnit*-Einheiten, um die als Reaktion auf ein Anklicken des Pfeilsymbols der Bildlaufleiste vertikal gerollt werden soll. Der voreingestellte Wert ist 1.

XPage `XPage: Integer;`

XPage ist die Anzahl von *XUnit*-Einheiten, um die in dem Bildlaufsymbol zur Verfügung stehenden Bereich als Reaktion auf ein Klicken horizontal gerollt werden soll. Per Voreinstellung ist *XPage* gleich der aktuellen Breite des Fensters in *XUnit*-Einheiten. Bei einer Änderung der Fenstergröße wird dieser Wert aktualisiert.

YPage `YPage: Integer;`

YPage ist die Anzahl von *YUnit*-Einheiten, um die in dem dem Bildlaufsymbol zur Verfügung stehenden Bereich als Reaktion auf ein Klicken vertikal gerollt werden soll. Per Voreinstellung

ist *YPage* gleich der aktuellen Höhe des Fensters in *YUnit*-Einheiten. Bei einer Änderung der Fenstergröße wird dieser Wert aktualisiert.

AutoMode

```
AutoMode: Boolean;
```

AutoMode ist *True,* wenn sich die Bildlaufleiste im automatischen Rollmodus befindet (*True* ist die Voreinstellung).

TrackMode

```
TrackMode: Boolean;
```

TrackMode ist *True*, wenn die Bildlaufleiste automatisch die Bewegungen des Bildlaufsymbols aktualisiert, indem das Fenster gerollt wird. Per Voreinstellung ist *TrackMode True.*

HasHScrollBar

```
HasHScrollBar: Boolean;
```

Wenn das zugehörige Fenster über eine horizontale Bildlaufleiste verfügt, ist *HasHScrollBar True.*

HasVscrollBar

```
HasVScrollBar: Boolean;
```

Wenn das zugehörige Fenster über eine vertikale Bildlaufleiste verfügt, ist *HasVScrollBar True.*

Methoden

Init

```
constructor Init(TheWindow: PWindow; TheXUnit,
TheYUnit: Integer; TheXRange, TheYRange: LongInt);
```

Initialisiert ein neues *TScroller*-Objekt mit *TheWindow* als dem besitzenden Fenster und *TheXUnit, TheYUnit, TheXRange*, und *TheYRange* jeweils als *XUnit, YUnit, XRange* und *YRange* und setzt *AutoMode* und *TrackMode* auf *True*. *HasHScrollBar* und *HasVScrollBar* werden abhängig von den Bildlaufleistenattributen des besitzenden Fensters gesetzt.

Load

```
constructor Load(var S: TStream);
```

Diese Methode initialisiert und lädt eine Bildlaufleiste aus dem Stream *S*, indem sie zuerst *TObject.Init* aufruft und dann die *TScroller*-Felder, ausgenommen *XPage, YPage, XPos* und *YPos*, liest.

AutoScroll

```
procedure AutoScroll; virtual;
```

Führt abhängig von der Position des Mauszeigers automatisches Rollen aus.

Querverweis*: TWindow.WMLButtonDown*

BeginView

```
procedure BeginView(PaintDC: HDC; var PaintInfo:
TPaintStruct); virtual;
```

Setzt den Ursprung des Bildschirmkontextes zum Neuzeichnen des besitzenden Fensters (*PaintDC*) gemäß der aktuellen Position des Bildlaufsymbols.

EndView

```
procedure EndView; virtual;
```

Aktualisiert die Positionen der Bildlaufsymbole des besitzenden Fensters, so daß sie in Übereinstimmung mit der Position von *TScroller* sind.

HScroll

```
procedure HScroll(ScrollRequest: Word; ThumbPos:
Integer); virtual;
```

Reagiert auf horizontale Bildlaufleistenereignisse durch Änderung der Position des Scrollers und des Bildlaufsymbols der horizontalen Bildlaufleiste.

Querverweis: *TWindow.WMHScroll*

IsVisibleRect

```
function IsVisibleRect(X,Y: LongInt; XExt, YExt:
Integer): Boolean;
```

Gibt *True* zurück, wenn ein Teil des durch die übergebenen Argumente bezeichneten Rechtecks gegenwärtig im besitzenden Fenster sichtbar ist.

ScrollBy

```
procedure ScrollBy(Dx, Dy: LongInt);
```

Rollt um den von *Dx* und *Dy* angegebenen Betrag. Aktualisiert außerdem die Darstellung des Fensters.

ScrollTo

```
procedure ScrollTo(X, Y: LongInt);
```

Rollt zu der Position, die durch *X* und *Y* vorgegeben ist. Aktualisiert außerdem die Darstellung des Fensters.

SetPageSize

```
procedure SetPageSize; virtual;
```

Setzt die Felder *XPage* und *YPage* entsprechend der aktuellen Breite und Höhe des Fensters.

Querverweis: *TWindow.WMSize*

SetRange

```
procdure SetRange(TheXRange, TheYRange: LongInt);
```

Diese Methode überschreibt die in dem Aufruf an *Init* übergebenen Werte *XRange* und *YRange*, indem sie diese mit *TheXRange* und *TheYRange* belegt. Dann ruft sie *SetSBarRange* auf, um den Bereich der Bildlaufleisten des besitzenden Fensters festzusetzen.

Querverweis: *TScroller.SetSBarRange*

SetSBarRange

```
procedure SetSBarRange; virtual;
```

Setzt den Bereich der Bildlaufleisten des besitzenden Fensters so, daß er mit dem Bereich von *TScroller* übereinstimmt.

Querverweis: *TWindow.SetupWindow*

SetUnits

```
procedure SetUnits(TheXUnit, TheYUnit: LongInt);
```

Setzt die Felder *XUnit* und *YUnit* auf *TheXUnit* bzw. *TheYUnit*.

Store
```
procedure Store(var S: TStream);
```
Diese Methode speichert den Scroller im Stream *S*, indem sie die *TScroller*-Felder zu ihm schreibt, ausgenommen sind die Felder *XPage, YPage, XPos* und *YPos*.

VScroll
```
procedure VScroll(ScrollRequest: Word; ThumbPos:
Integer); virtual;
```
Reagiert auf vertikale Bildlaufleistenereignisse mit Änderung der Position des Scrollers und des Bildlaufsymbols der vertikalen Bildlaufleiste.

Querverweis: *TWindow.WMVScroll*

TSortedCollection **WObjects**

Beschreibung *TSortedCollection* ist ein Abkömmling von *TCollection,* der Kollektionen implementiert, die nach einem Schlüssel ohne Duplikate sortiert sind. Die Sortierordnung wird durch eine virtuelle Methode *TSortedCollection.Compare* definiert, die Sie mit Ihrer eigenen Definition überschreiben müssen. Sobald neue Elemente hinzugefügt werden, werden sie automatisch in der Reihenfolge eingefügt, die die Methode *Compare* vorgibt. Elemente können mittels der binären Suchmethode *TSortedCollection.Search* aufgefunden werden.

Die virtuelle Methode *KeyOf,* die für *Compare* einen Zeiger zurückgibt, kann ebenfalls überschrieben werden, wenn *Compare* zusätzliche Informationen benötigt.

Felder

Duplicates
```
Duplicates: Boolean;
```
Das Feld *Duplicates* legt fest, ob Elemente mit gleichem Schlüssel akzeptiert werden.

Wenn *Duplicates True* ist, ist es möglich, Einträge mit gleichem Schlüssel in die Kollektion einzufügen. Wenn *Duplicates False* ist (dies ist die Voreinstellung), *ersetzt* ein neues Element mit gleichem Schlüssel das mit diesem Schlüssel vorhandene Element.

Methoden

Load
```
constructor Load(var S: TStream);
```
Diese Methode initialisiert und lädt eine sortierte Kollektion aus dem Stream *S*, indem sie zuerst *TCollection.Load* aufruft und dann das von *TSortedCollection* hinzugefügte Feld *Duplicates* liest.

Querverweis: *TCollection.Load*

Compare

```
function Compare(Key1, Key2: Pointer): Integer; virtual;
```

Compare ist eine abstrakte Methode, die in allen abgeleiteten Typen überschrieben werden muß. *Compare* vergleicht zwei Schlüsselwerte miteinander und gibt dann ein Ergebnis wie das folgende zurück:

-1	wenn *Key1* < *Key2*
0	wenn *Key1* = *Key2*
1	wenn *Key1* > *Key2*

*Key*1 und *Key2* sind Zeigerwerte, die mit der Methode *TSortedCollection.KeyOf* aus ihren zugehörigen Kollektionselementen extrahiert werden. Die Methode *TSortedCollection.Search* implementiert einen binären Suchvorgang durch die Kollektionselemente, die dabei *Compare* zum Vergleich der Elemente benutzt.

Querverweis: *TSortedCollection.KeyOf, TSortedCollection.Compare*

IndexOf

```
function IndexOf(Item: Pointer): Integer; virtual;
```

Verwendet *TSortedCollection.Search*, um den Index des gegebenen Elements *Item* aufzufinden. Wenn das Element nicht in der Kollektion ist, liefert *IndexOf* -1 zurück.

TSortedCollection.IndexOf ist wie folgt implementiert:

```
if Search(KeyOf(Item), I)     then  IndexOf := I
else IndexOf := -1;
```

Querverweis: *TSortedCollection.Search*

Insert

```
procedure Insert(Item: Pointer); virtual;
```

Wenn das Zielelement in der sortierten Kollektion nicht gefunden wird, wird es an der richtigen Indexposition eingefügt. Die Methode ruft *TSortedCollection.Search* auf, um festzustellen, ob das Element existiert bzw. wo es eingefügt werden soll, wenn es nicht existiert. *TSortedCollection.Insert* ist wie folgt implementiert:

```
if not  Search(KeyOf(Item), I)     then  AtInsert(I, Item);
```

Querverweis: *TSortedCollection.Search*

KeyOf

```
function KeyOf(Item: Pointer): Pointer; virtual;
```

Für das gegebene Element *Item* aus der Kollektion sollte *KeyOf* den entsprechenden Schlüssel des Elements zurückliefern. Die voreingestellte Methode *TSortedCollection.KeyOf* gibt *Item* zurück. *KeyOf* wird in den Fällen überschrieben, in denen der Schlüssel des Elements nicht das Element selbst ist.

Querverweis: *TSortedCollection.IndexOf*

Search

```
function Search(Key: Pointer; var Index: Integer): Boolean; virtual;
```

Wenn das von *Key* bezeichnete Element in der sortierten Kollektion gefunden wurde, gibt die Methode *True* zurück und *Index* wird auf den gefundenen Index gesetzt. Andernfalls wird *Index* auf den Index gesetzt, unter dem das Element eingefügt würde.

Querverweis: *TSortedCollection.Compare, TSortedCollection.Insert*

Store

```
procedure Store(var S: TStream);
```

Diese Methode speichert die sortierte Kollektion und alle ihre Elemente im Stream *S*. Hierzu ruft sie *TCollection.Store* auf, um die Kollektion zu schreiben, und schreibt dann das Feld *Duplicates* in den Stream.

Querverweis: *TCollection.Store*

TStatic

WObjects

Beschreibung

TStatic ist ein Schnittstellenobjekt, das für ein entsprechendes Textfeld steht. *TStatic*-Objekte werden normalerweise nicht in Dialogfenstern (*TDialog* oder *TDlgWindow*) verwendet, aber sie werden eingesetzt, wenn man ein selbständiges statisches Textfeld im Client-Bereich eines Fensters darstellen will.

Felder

TextLen

```
TextLen: Word;
```

TextLen enthält die Größe des Textpuffers für statische Felder. Die Anzahl der Zeichen, die tatsächlich im Textfeld gespeichert werden kann, ist um 1 kleiner als *TextLen*, da das abschließende Nullzeichen nicht mitgezählt wird. *TextLen* ist auch die Anzahl der Zeichen, die von der Methode *Transfer* transferiert wurde.

Methoden

Init

```
constructor Init(AParent: PWindowsObject; AnID: Integer; ATitle: PChar; X, Y, W, H: Integer; ATextLen: Word);
```

Diese Methode initialisiert ein statisches Textfeldobjekt für das übergebene übergeordnete Fenster (*AParent*) mit der Steuerelement-ID (*AnID*), dem Text (*ATitle*), der Position relativ zum Ursprung des Client-Bereichs des übergeordneten Fensters (*X*,*Y*), der Breite (*W*), der Höhe (*H*) und der Textlänge (*ATextLen*). Das statische Textfeld ist linksbündig, da *TStatic.Init* den Windows-Stil *ss_Left* zum Feld *Attr.Style* des Objekts hinzufügt.

Außerdem löscht diese Methode den Stil *ws_TabStop* und ruft dann per Voreinstellung *DisableTransfer* auf, um die *TStatic*-Objekte aus dem Transfermechanismus auszuschließen.

InitResource

```
constructor InitResource(AParent: PWindowsObject;
ResourceID: Word; ATextLen: Word);
```

Ordnet einem *TStatic*-Objekt die Ressource des statischen Textfeldes zu, die von *ResourceID* bezeichnet wird und setzt das Feld *TextLen* auf *ATextLen*. Außerdem ruft sie *TControl.InitResource* auf, um das Objekt zu initialisieren und zuzuordnen.

Querverweis: *TControl.InitResource*

Load

```
constructor Load(var S: TStream);
```

Diese Methode initialisiert und lädt ein statisches Textfeld aus dem Stream *S,* indem sie *TControl.Load* aufruft und dann das Feld *TextLen* liest.

Querverweis: *TWindow.Load*

Clear

```
procedure Clear;
```

Löscht den Text des statischen Textfeldes.

GetClassName

```
function GetClassName: PChar; virtual;
```

Gibt »Static« – die Bezeichnung der Fensterklasse von *TStatic* – zurück.

GetText

```
function GetText(ATextString: PChar, MaxChars:
Integer): Integer;
```

Ruft den Text des statischen Feldes ab und speichert ihn im Argument *ATextString*.

MaxChars bezeichnet die Maximalgröße von *ATextString*. *GetString* gibt die Größe des abgerufenen Strings zurück.

SetText

```
procedure SetText(ATextString: PChar);
```

Diese Methode setzt für den Text des statischen Feldes den in *ATextString* übergebenen String ein.

Store

```
procedure Store(var S: TStream);
```

Diese Methode speichert das Feld im Stream *S*, indem sie *TControl.Store* aufruft und das Feld *TextLen* in den Stream schreibt.

Querverweis: *TWindow.Store*

Transfer

```
function Transfer(DataPtr: Pointer; TransferFlag:
Word): Word; virtual;
```

Diese Methode transferiert *TextLen* Zeichen aus dem aktuellen Text des Textfeldes zu oder aus der Speicherstelle, auf die

DataPtr zeigt. Ist *TransferFlag tf_GetData*, wird der Text an die Speicherstelle übertragen. Ist *TransferFlag tf_SetData*, wird der Text des statischen Feldes durch den Text an der Speicherstelle ersetzt. *Transfer* gibt *TextLen* – die Anzahl der aus der Speicherstelle abgerufenen oder dort gespeicherten Daten – zurück. Ist *TransferFlag tf_SizeData*, gibt *Transfer* die Größe der Transferdaten zurück.

TStream

WObjects

Beschreibung

TStream ist ein allgemeines abstraktes Objekt, das polymorphe Ein- und Ausgabe zu oder von einem Speichermedium bietet. Sie können Ihre eigenen abgeleiteten Stream-Objekte erzeugen, indem Sie die virtuellen Methoden überschreiben: *GetPos*, *GetSize*, *Read*, *Seek*, *Truncate* und *Write*.
ObjectWindows selbst macht das, um *TDosStream* und *TEmsStream* abzuleiten. Für gepufferte abgeleitete Streams müssen Sie auch *TStream.Flush* überschreiben

Felder

Status

```
Status: Integer;
```

Kennzeichnet den aktuellen Status des Streams wie in der folgenden Tabelle dargestellt.

Wenn *Status* nicht *stOk* ist, werden alle Operationen im Stream unterbrochen bis *Reset* aufgerufen wird.

Fehlercode	Bedeutung
stOk	Kein Fehler
stError	Zugriffsfehler
stInitError	Stream kann nicht initialisiert werden
stReadError	Leseversuch über das Ende des Streams hinaus
stWriteError	Stream kann nicht erweitert werden
stGetError	*Get* für einen unregistrierten Objekttyp
stPutError	*Put* für einen unregistrierten Objekttyp

ErrorInfo

```
ErrorInfo: Integer;
```

Wenn *Status* nicht *stOk* ist, enthält *ErrorInfo* zusätzliche Information. *ErrorInfo* enthält für die *Status*-Werte *stError, stInitError, stReadError*, *stWriteError* und – falls verfügbar – den DOS- oder EMS-Fehlercode.

Ist *Status stGetError*, enthält *ErrorInfo* die Objekttypen-ID (das Feld *ObjType* eines *TStreamRec*) des unregistrierten Objekttyps. Ist *Status stPutError*, enthält *ErrorInfo* den VMT-Datensegment-Offset (das Feld *VmtLink* eines *TStreamRec*) des unregistrierten Objekttyps

Methoden

CopyFrom

```
procedure CopyFrom(var S: TStream; Count: LongInt);
```

Diese Methode kopiert *Count* Bytes aus dem Stream *S* an das aufrufende Stream-Objekt. Beispiel:

```
{ Erzeugt eine Kopie des gesamten Streams }
NewStream := New(TEmsStream, init(OldStream^.GetSize));
OldStream^.Seek(0);
NewStream^.CopyFrom(OldStream, OldStream^.GetSize);
```

Querverweis: *TStream.GetSize, TObject.Init*

Error

```
procedure Error(Code, Info: Integer); virtual;
```

Diese Methode wird immer aufgerufen, wenn ein Stream-Fehler auftritt. Die Standardmethode *TStream.Error* speichert *Code* und *Info* in den Feldern *Status* und *ErrorInfo* und ruft dann die Prozedur auf, auf die die globale Variable *StreamError* zeigt, wenn diese nicht **nil** ist. Sobald ein Fehler aufgetreten ist, werden alle Stream-Operationen im Stream unterbrochen, bis *Reset* aufgerufen wird.

Querverweis: *TStream.Reset,* Variable *StreamError*

Flush

```
procedure Flush; virtual;
```

Diese abstrakte Methode muß überschrieben werden, wenn der von Ihren definierte Nachkomme einen Puffer implementiert. Diese Methode kann jeden Puffer entleeren, indem sie den Lesepuffer löscht und/oder den Schreibepuffer in den Stream schreibt. Per Voreinstellung wird von *TStream.Flush* nichts ausgeführt.

Querverweis: *TBufStream.Flush*

Get

```
function Get: PObject;
```

Diese Methode liest ein Objekt aus dem Stream, das vorher mit *TStream.Put* in den Stream geschrieben worden sein muß. *Get* liest zuerst eine Objekttyp-ID (ein Word) aus dem Stream und findet dann den zugehörigen Objekttyp durch Vergleich der ID mit dem *ObjType*-Feld aller registrierten Objekttypen (vgl. *TStreamRec*-Typ). Anschließend ruft sie den Konstruktor *Load* des Objekttyps auf, um das Objekt zu erzeugen und zu laden. Ist die vom Stream gelesene Objekttyp-ID 0, gibt *Get* einen **nil**-Zeiger zurück. Wenn die Objekttyp-ID nicht mit *RegisterType* registriert wurde, ruft *Get TStream.Error* auf und gibt einen **nil**-Zeiger zurück, andernfalls gibt *Get* einen Zeiger auf das neu erzeugte Objekt zurück.

Querverweis: *TStream.Put, RegisterType, TStreamRec, Load*-Methoden

GetPos

```
function GetPos: LongInt; virtual;
```

GetPos ist eine abstrakte Methode, die überschrieben werden muß. Sie gibt den Wert der aktuellen Position des aufrufenden Streams zurück.

Querverweis: *TStream.Seek*

GetSize

```
function GetSize: LongInt; virtual;
```

GetSize ist eine abstrakte Methode, die überschrieben werden muß. Sie gibt die Gesamtgröße des aufrufenden Streams zurück.

Put

```
procedure Put(P: PObject);
```

Diese Methode schreibt ein Objekt in den Stream. Dieses kann mit *TStream.Get* später wieder aus dem Stream gelesen werden. *Put* findet zuerst durch Vergleich des VMT-Offsets des Objekts mit dem *VmtLink*-Feld aller registrierten Objekttypen (vgl. *TStreamRec*-Typ) den Record zur Typenregistrierung des Objekts. Danach schreibt sie die Objekttypen-ID (das *ObjType*-Feld des Registrierungsrecords) in den Stream und ruft schließlich die Methode *Store* dieses Objekttyps auf, um das Objekt zu schreiben.

Ist das an *Put* übergebene Argument *P* **nil**, schreibt *Put* ein Word mit dem Wert 0 in den Stream. Wenn der Objekttyp von *P* nicht mit *RegisterType* registriert wurde, ruft *Put TStream.Error* auf und schreibt nichts in den Stream.

Querverweis: *TStream.Get, Registertype, TStreamRec, Store*-Methoden

Read

```
procedure Read(var Buf; Count: Word); virtual;
```

Read ist eine abstrakte Methode, die in allen abgeleiteten Typen überschrieben werden muß. Sie sollte *Count* Bytes aus dem Stream in *Buf* einlesen und dann die aktuelle Position des Streams um *Count* Bytes erhöhen. Tritt ein Fehler auf, sollte *Read Error* aufrufen und *Buf* mit *Count* Bytes mit dem Wert 0 auffüllen.

Querverweis: *TStream.Write, TStream.Error.*

ReadStr

```
function ReadStr: PString;
```

Diese Methode liest einen String von der aktuellen Position des aufrufenden Streams und gibt einen *PString*-Zeiger zurück. *TStream.ReadStr* ruft *GetMem* auf, um (Stringlänge+1) Bytes für den String zu reservieren.

Querverweis: *TStream.WriteStr*

Reset

```
procedure Reset;
```

Diese Methode setzt jede Stream-Fehlerbedingung zurück, indem sie *Status* und *ErrorInfo* auf 0 setzt. Mit Hilfe dieser Methode können Sie die Streamverarbeitung nach einer von Ihnen behobenen Fehlerbedingung wieder aufnehmen.

Querverweis: *TStream.Status*, *TStream.ErrorInfo*, *stXXXX*-Fehlercodes

Seek

```
procedure Seek(Pos: LongInt); virtual;
```

Dies ist eine abstrakte Methode, die von allen Nachkommen überschrieben werden muß.

TStream.Seek setzt die aktuelle Position auf *Pos* Bytes ab dem Beginn des aufrufenden Streams. Der Startpunkt ist an Position 0.

Querverweis: *TStream.GetPos*

StrRead

```
function StrRead: PChar;
```

Diese Methode liest einen String aus dem Stream, indem sie zuerst die Länge des Strings ermittelt und dann diese Anzahl von Zeichen einliest. Gibt einen Zeiger auf den gelesenen nullterminierten String zurück.

Querverweis: *TStream.StrWrite*

StrWrite

```
procedure StrWrite(P: PChar);
```

Diese Methode schreibt den nullterminierten String *P* in den Stream, indem sie zuerst die Stringlänge und dann diese Anzahl von Zeichen schreibt.

Querverweis: *TStream.StrRead*

Truncate

```
procedure Truncate; virtual;
```

Dies ist eine abstrakte Methode, die in allen abgeleiteten Typen überschrieben werden muß.

TStream.Truncate löscht alle Daten aus dem aufrufenden Stream von der aktuellen Position bis zum Ende.

Querverweis: *TStream.GetPos*, *TStream.Seek*

Write

```
procedure Write(var Buf; Count: Word); virtual;
```

Dies ist eine abstrakte Methode, die in allen abgeleiteten Typen überschrieben werden muß.

Write sollte *Count* Byte aus *Buf* in den Stream schreiben und die aktuelle Position des Streams um *Count* Bytes erhöhen. Wenn ein Fehler auftritt, sollte *Write Error* aufrufen.

Querverweis: *TStream.Read, TStream.Error*.

WriteStr

```
procedure WriteStr(P: PString);
```

Schreibt den String *P*^, beginnend an der aktuellen Position, in den aufrufenden Stream.

Querverweis: *TStream.ReadStr*

TStrCollection — WObjects

Beschreibung

TStrCollection ist ein einfacher Abkömmling von *TSortedCollection*, der eine sortierte Liste von ASCII-Strings implementiert. Die Methode *TStrCollection.Compare* ist überschrieben, um die konventionelle lexikographische Anordnung von ASCII-Strings zu erhalten. Sie können *Compare* überschreiben, um andere Sortierordnungen zu realisieren, wie zum Beispiel für nicht-englische Zeichensätze.

Methoden

Compare

```
function Compare(Key1, Key2: Pointer): Integer; virtual;
```

Vergleicht die Strings *Key1*^ und *Key2*^ folgendermaßen: Ist *Key1* < *Key2*, wird -1, ist *Key1* = *Key2*, wird 0 und ist *Key1* > *Key2*, wird +1 zurückgegeben.

Querverweis: *TSortedCollection.Search*

FreeItem

```
procedure FreeItem(Item: Pointer); virtual;
```

Löscht den String *Item*^ aus der sortierten Kollektion und gibt ihn frei.

GetItem

```
function GetItem(var S: TStream): Pointer; virtual;
```

Liest per Voreinstellung durch Aufruf von *S.ReadStr* einen String aus dem Stream *TStream*

Querverweis: *TStream.ReadStr*

PutItem

```
procedure PutItem(var S: TStream; Item: Pointer);
virtual;
```

Schreibt per Voreinstellung durch Aufruf von *S.WriteStr* den String *Item*^ in den Stream *TStream*.

Querverweis: *TStream.WriteStr*

TWindow — WObjects

Beschreibung

TWindow definiert das grundlegende Verhalten für alle Fenster und Steuerelementobjekte. Objektinstanzen von *TWindow* sind leere generische Fenster, aber sie können Menüs, Mauszeigerformen und Symbole definieren.

Felder

Attr

```
Attr: TWindowAttr;
```

Attr enthält einen *TWindowAttr*-Record, der die Fenstererzeugungsattribute definiert. Das sind die Merkmale, die die Erzeugung des zugehörigen Schnittstellenelements eines Fensterobjekts beeinflussen. Das schließt den dem Fenster zugeordneten Text, Stil, erweiterten Stil, die Position und Größe, das Fenster-Handle und die Steuerelement-ID ein. Diese Attribute werden normalerweise im *Init*-Konstruktor des Objekts mit Standardwerten belegt, sie können aber im *Init*-Konstruktor eines Nachkommens des Typs überschrieben werden. Hier ist die Record-Definition für *TWindowAttr:*

```
TWindowAttr =    record
  Title: PChar;
  Style: LongInt;
  ExStyle: LongInt;
  X, Y, W, H: Integer;
  Param: Pointer;
   case  Integer  of
    0: (Menu: HMenu); { Handle des Fenstermenis }
    1: (Id: Integer); { Bezeichner des Steuerelements }
end;
```

DefaultProc

```
DefaultProc: TFarProc;
```

Diese Methode enthält die Adresse der Standardfensterprozedur, die die standardmäßige Windows-Verarbeitung von Windows-Botschaften definiert.

FocusChild-Handle

```
FocusChildHandle: THandle;
```

FocusChildHandle speichert das Windows-Handle zu dem untergeordneten Fenster, das bei der letzten Aktivierung des Fensters den Fokus hatte.

Scroller

```
Scroller: PScroller;
```

Scroller enthält einen Zeiger auf ein *TScroller*-Objekt, das der Scroller des Fensters ist, wenn ein solcher vorhanden ist. Ein Scroller sollte im Konstruktor *Init* des Fensters initialisiert werden.

Methoden

Init

```
constructor Init(AParent: PWindowsObject; ATitle:
PChar);
```

Diese Methode initialisiert ein Fensterobjekt für das in *AParent* übergebene übergeordnete Fenster und mit dem in *ATitle* übergebenen zugehörigen Text (bei Fenstern ist dies die Über-

schrift). *AParent* sollte für Hauptfenster **nil** sein, weil diese keine übergeordneten Fenster haben. Wenn das Fenster kein untergeordnetes MDI-Fenster ist, wird das *Attr.Style*-Feld auf *ws_OverlappedWindow* gesetzt, ansonsten auf *ws_ClipSiblings*. Die Positions- und Abmessungsfelder im *Attr*-Record werden auf geeignete Standardwerte für überlappte und Pop-up-Fenster gesetzt. Sie können *TWindow.Init* in den von Ihnen abgeleiteten Typen überschreiben. Hierzu müssen Sie zuerst explizit *TWindow.Init* aufrufen, anschließend können Sie den *Attr*-Feldern neue Werte zuweisen. *Scroller* wird per Voreinstellung auf **nil** gesetzt.

InitResource

```
constructor InitResource(AParent: PWindowsObject;
ResourceID: Word);
```

Diese Methode initialisiert ein ObjectWindows-Objekt, das einem mit einer Ressourcendefinition erzeugten Schnittstellenelement (üblicherweise einem Steuerelement) zugeordnet werden soll. *InitResource* ruft *TWindowsObject.Init* auf.

Querverweis: *TWindowsObject.Init*

Load

```
constructor Load(var S: TStream);
```

Diese Methode initialisiert und lädt ein Fenster aus dem Stream *S*, indem sie zuerst *TWindowsObject.Load* aufruft und dann die von *TWindow* hinzugefügten Felder (*Attr* und *Scroller*) liest.

Querverweis*: TWindowsObject.Load*

Done

```
destructor Done; virtual;
```

Diese Methode gibt das *TScroller*-Objekt in *Scroller* frei – falls dieses vorhanden ist – und ruft dann *TWindowsObject.Done* auf, um das gesamte Objekt freizugeben.

Create

```
function Create: Boolean; virtual;
```

Diese Methode erzeugt das zugehörige Schnittstellenelement des Fensterobjekts, wenn das Objekt nicht mit *InitResource* initialisiert wurde, da in diesem Fall das Schnittstellenelement bereits vorhanden ist. Für noch nicht registrierte Fenster ruft *Create Register* auf, erzeugt anschließend das Fenster und ruft *SetupWindow* auf, das Sie zum Inititalisieren des neu erzeugten Fensters definieren können. Normalerweise geschieht dies durch Erzeugen untergeordneter Fenster und Zeichnen von Graphik oder Text. *Create* gibt bei erfolgreicher Ausführung *True* zurück, ansonsten ist der Rückgabewert *False* und *Create* ruft *Error* auf.

Normalerweise rufen Sie *Create* niemals direkt auf. *Create* wird von *TApplication.MakeWindow* aufgerufen, die zuerst eine Prüfung auf ausreichenden Speicherplatz durchführt.

Querverweis: *TApplication.MakeWindow, TWindowsObject.Register, TWindowsObject.SetupWindow, TWindow.InitResource, TWindowsObject.Error*

DefWndProc

```
procedure DefWndProc(var Msg: TMessage); virtual;
```

Diese Methode ruft die Standardprozedur des Fensters auf, die die Standardverarbeitung für eintreffende Windows-Botschaften durchführt. Sie speichert das Ergebnis dieses Aufrufs im *Result*-Feld des Botschaft-Records *Msg*.

Destroy

```
procedure Destroy; virtual;
```

Ruft nur *TWindowsObject.Destroy* auf, um das Fensterelement freizugeben und zu entfernen, außer es handelt sich um ein untergeordnetes MDI-Fenster. In diesem Fall sendet *Destroy* für eine geeignete Standardverarbeitung eine *wm_MDIDestroy*-Botschaft an das zugehörige MDI-Client-Fenster.

GetId

```
function GetId: Integer; virtual;
```

GetId gibt den Fensterbezeichner – wie etwa eine Steuerelement-ID – zurück.

GetWindowClass

```
procedure GetWindowClass(var AWndClass: TWndClass);
virtual;
```

Füllt einen in *AWndClass* übergebenen Fensterklassen-Record mit Standardwerten für seine Registrierungsattribute. Das Stil-Feld wird mit *cs_HRedraw* oder *cs_VRedraw* belegt, das Symbol als generisches Symbol und der Mauszeiger als normaler Pfeilzeiger definiert. Die Hintergrundfarbe wird auf die Fensterhintergrundfarbe des Systems gesetzt. Die zu registrierende Klassenbezeichnung wird über einen Aufruf von *GetClassName* ermittelt.

Querverweis: *TWindowsObject.GetClassName, TWindowsObject.Register, TWindow.Create*

Paint

```
procedure Paint(PaintDC: HDC; var PaintInfo:
TPaintStruct); virtual;
```

Dient als Platzhalter für abgeleitete Typen, die *Paint*-Methoden definieren. Diese Methode wird automatisch als Reaktion auf eine Aufforderung von Windows (*wm_Paint*) aufgerufen, den Fensterinhalt neu darzustellen.

PaintDC muß als Bildschirmkontext benutzt werden. Er wird schon vor dem Aufruf an *Paint* eingerichtet und nach *Paint* wieder freigegeben. Die übergebene Zeichenstruktur *PaintInfo* enthält genauere Informationen über die Zeichenanforderung.

Querverweis: *TWindow.WMPaint*

SetupWindow `procedure SetupWindow; virtual;`

Richtet das neu erzeugte Fenster ein. Ist das Fenster ein untergeordnetes MDI-Fenster, ruft *SetupWindow SetFocus* auf, um dem neuen Fenster den Fokus zuzuteilen. Wenn das Fenster ein Scroller-Objekt hat, ruft *SetupWindow SetSBarRange* auf, um den Bereich seiner Bildlaufleisten festzulegen.

Querverweis: *TScroller.SetSBarRange*

Store `procedure Store(var S: TStream);`

Diese Methode speichert das Fenster im Stream *S*, indem sie zuerst *TWindowsObject.Store* aufruft und dann die von *TWindow* hinzugefügten Felder (*Attr* und *Scroller*) in den Stream schreibt.

Querverweis: *TWindowsObject.Store*

WmActivate `procedure WMActivate(var Msg: TMessage); virtual wm_First + wm_Activate;`

Diese Methode wird für Fenster verwendet, die Tastaturbotschaften für ihre Steuerelemente entgegennehmen.

Sie reagiert auf Änderungen des Fokus, indem sie die Handles des untergeordneten Steuerelements, das gerade den Fokus hat, in *FocusChildHandle* abspeichert und den Fokus wiederherstellt.

Querverweis: *TWindowsObject.EnableKBHandler*

WMCreate `procedure WMCreate(var Msg: TMessage); virtual; wm_First + wm_Create;`

Reagiert auf die *wm_Create*-Botschaft mit Aufruf von *SetupWindow* und anschließendem Aufruf von *DefWndProc*. Da die Fenstererzeugung unter ObjectWindows anders als unter Windows gehandhabt wird, muß die *wm_Create*-Botschaft abgefangen und dazu benutzt werden, Fensterobjektattribute einzurichten.

Querverweis: *TWindow.SetupWindow*

WMDestroy `procedure WMDestroy(var Msg: TMessage); virtual wm_First + wm_Destroy;`

Reagiert auf das Schließen des Fensters normalerweise mit einem Aufruf von *TWindowsObject.WMDestroy*, um das Fensterelement zu entfernen und freizugeben. Handelt es sich jedoch um ein untergeordnetes MDI-Fenster, setzt *WMDestroy* das Handle des Fensters *HWindow* auf 0 und ruft die *RemoveChild*-Methode des MDI-Rahmenfensters auf.

Querverweis: *TWindowsObject.RemoveChild*

WMHScroll

```
procedure WMHScroll(var Msg: TMessage); virtual
wm_First + wm_HScroll;
```

Für Fenster mit Scrollern. Reagiert auf horizontale Bildlaufleistenereignisse durch den Aufruf der Methode *HScroll* des Scrollers und den Aufruf von *DefWndProc*.

Querverweis*: TScroller.HScroll*

WMLButton-Down

```
procedure WMLButtonDown(var Msg: TMessage); virtual
wm_First + wm_LButtonDown;
```

Für Fenster mit automatischem Rollmechanismus. Reagiert auf das Klicken der linken Maustaste, indem sie solange die gesamte weitere Mauseingabe abfängt, bis die Maustaste wieder losgelassen wird. Wenn Sie diese Methode zur eigenen Verarbeitung von Mausklicks überschreiben, aber trotzdem automatisches Rollen einsetzen möchten, rufen Sie diese Methode aus Ihrer *WMLButtonDownDown*-Methode heraus auf.

WMPaint

```
procedure WMPaint(var Msg: TMessage); virtual wm_First
+ wm_Paint;
```

Reagiert auf die Windows-Botschaft *wm_Paint* durch Aufruf der *Paint*-Methode des Fensterobjekts. Verfügt das Fenster über einen Scroller, ruft *WMPaint* vor dem Aufruf von *Paint BeginView* und danach *EndView* auf.

Querverweis: *TWindow.Paint, TScroller.BeginView*, *TScroller.EndView*

WMSize

```
procedure WMSize(var Msg: TMessage); virtual wm_First +
wm_Size;
```

Für Fenster mit Scrollern. Reagiert auf eine Änderung der Fenstergröße mit einem Aufruf von *SetPageSize*.

Querverweis: *TScroller.SetPageSize*

WMVScroll

```
procedure WMVScroll(var Msg: TMessage); virtual
wm_First + wm_VScroll;
```

Für Fenster mit Scrollern. Reagiert auf vertikale Bildlaufereignisse durch Aufruf der Methode *VScroll* des Scrollers und durch Aufruf von *DefWndProc*.

Querverweis: *TScroller.VScroll*

TWindowsObject **WObjects**

Beschreibung *TWindowsObject* definiert die grundlegenden Verhaltensweisen und Eigenschaften aller Schnittstellenobjekte, einschließlich Fenstern, Dialogfenstern und Steuerelementen.

TWindowsObject ist ein abstrakter Objekttyp, seine Metho-

den sind nur für abgeleitete Typen sinnvoll. Diese implementieren das grundlegende Konzept der Erzeugung und Entfernung von Schnittstellenelementen sowie der Fensterklassenregistrierung und den Mechanismus zur Beantwortung von Botschaften.

Felder

Childlist

```
ChildList: PWindowsObject;
```

ChildList ist eine verkettete Liste aller untergeordneten Fensterobjekte des Schnittstellenobjekts. Dazu gehören Pop-up-Fenster, Dialogfenster und Steuerelemente. *ChildList* zeigt immer auf das zuletzt hinzugefügte Objekt.

Flags

```
Flags: Byte;
```

Flags ist ein Datenbyte, dessen Bits dazu verwendet werden, die folgenden Fensterattribute zu speichern: Tastaturverarbeitung, automatische Erzeugung, Transfer, MDI-Status und die Ressourcenerzeugung. *Flags* enthält eine oder mehrere der *wb_*-Konstanten, die in Kapitel 6 »Außerhalb der Objekte« dokumentiert sind.

Querverweis: *TWindowsObject.SetFlags, TWindowsObject.-IsFlagSet*

HWindow

```
HWindow.HWnd;        only                Read
```

HWindow enthält ein Handle zum zugehörigen Schnittstellenelement. Wenn es kein solches Element gibt, ist *HWindow* gleich 0 (der Wert eines ungültigen Handles). Bei der Erzeugung des zugeordneten Schnittstellenelements (*Create*) wird *HWindow* mit einem neuen Handle belegt. Bei der Entfernung des zugeordneten Schnittstellenelements (*WMNCDestroy*) wird *HWindow* auf 0 gesetzt.

Instance

```
Instance: TFarProc;
```

Instance zeigt auf den Code, der vor dem Eintritt in die gemeinsam benutzte Fensterprozedur von ObjectWindows ausgeführt wird.

Parent

```
Parent: PWindowsObject;
```

PWindowsObject zeigt auf das Schnittstellenobjekt, das als das übergeordnete Fenster dieses Schnittstellenobjekts dient. Zum Beispiel könnte das Feld *Parent* eines Steuerelementobjekts, das in seinem übergeordneten Fenster erscheint, auf dessen Fensterobjekt zeigen.

Status

```
Status: Integer;
```

Status kennzeichnet den Erfolg des aktuellen Versuchs, ein Schnittstellenobjekt und seine zugeordneten Schnittstellen-

elemente zu initialisieren. Der Prozeß ist erfolgreich, solange *Status* größer oder gleich 0 ist, jedoch nicht, wenn er negativ ist. Nachkommen von *TWindowsObject* – einschließlich *TWindow* und *TDialog* – überprüfen *Status*, bevor sie ihre zugeordneten Elemente erzeugen. Benutzen Sie *Status* im Code der von Ihnen abgeleiteten Typen, um einen Initialisierungsfehler anzuzeigen.

Mögliche Fehlerwerte sind: *em_InvalidWindow, em_InvalidClient, em_InvalidChild* und *em_InvalidMainWindow*.

TransferBuffer

```
TransferBuffer: Pointer;
```

TransferBuffer zeigt auf einen Transfer-Record, der von einer Anwendung definiert wurde, die den Transfermechanismus benutzt. Andernfalls ist es **nil**.

Methoden

Init

```
constructor Init(AParent: PWindowsObject);
```

Initialisiert das Schnittstellenobjekt. Konstruktoren abgeleiteter Typen müssen mit einem Aufruf von *TWindowsObject.Init* enden. *Init* ruft *EnableAutoCreate* auf, so daß das untergeordnete Fenster standardmäßig zusammen mit seinen übergeordneten Fenstern erzeugt und dargestellt wird, und fügt außerdem das Schnittstellenobjekt zur Liste der untergeordneten Fenster seines übergeordneten Fensterobjekts hinzu.

Querverweis : *TWindowsObject.EnableAutoCreate, TWindowsObject.AddChild*

Load

```
constructor Load(var S: TStream);
```

Diese Methode initialisiert und lädt ein Schnittstellenobjekt aus dem Stream *S*, indem sie den *Status*, andere Attribute und die Größe von *ChildList* liest und anschließend jedes untergeordnete Fenster lädt.

Done

```
Destructor Done; virtual;
```

Diese Methode entfernt das Schnittstellenobjekt, indem sie zuerst das zugehörige Schnittstellenelement entfernt – falls vorhanden – und dann *TObject.Done* aufruft. Sie gibt alle seine untergeordneten Fenster frei und löscht sich selbst aus der Liste untergeordneter Fenster seines übergeordneten Fensters. Destruktoren abgeleiteter Typen müssen mit einem Aufruf von *TWindowsObject.Done* enden.

CanClose

```
function CanClose: Boolean; virtual;
```

Ruft die Methode *CanClose* für jedes untergeordnete Fenster auf und gibt *False* zurück, wenn eines der untergeordneten Fenster *False* zurückgibt, um anzuzeigen, daß es nicht angebracht ist, das Schnittstellenelement zu schließen. Wenn alle

CanClose-Methoden der untergeordneten Fenster *True* zurückgeben, gibt *CanClose True* zurück.

ChildWithId

```
function ChildWithId(Id: Integer): PWindowsObject;
```

Diese Methode gibt einen Zeiger auf das Fenster in der Liste untergeordneter Fenster zurück, das die übergebene ID hat. Wird kein passendes untergeordnetes Fenster gefunden, gibt *ChildWithId* **nil** zurück. Wenn *Id* -1 ist, gibt sie das erste Nicht-Steuerelementobjekt in der Liste untergeordneter Fenster zurück.

Create

```
function Create: Boolean; virtual;
```

Dies ist eine abstrakte Methode, die in den abgeleiteten Typen überschrieben wird, um das zugehörige Schnittstellenelement des Schnittstellenobjekts zu erzeugen.

DefChildProc

```
procedure DefChildProc(var Msg: TMessage); virtual;
```

Diese Methode führt die Standardverarbeitung für eine eintreffende Botschaft aus, die auf einer ID untergeordneter Fenster basiert, indem sie das Feld *Result* von *Msg* auf 0 setzt und damit anzeigt, daß die Botschaft nicht verarbeitet wurde.

DefCommandProc

```
procedure DefCommandProc(var Msg: TMessage); virtual;
```

Führt die Standardverarbeitung für eintreffende befehlsbasierende Botschaften aus, indem sie das Feld *Result* von *Msg* auf 0 setzt und so anzeigt, daß die Botschaft nicht verarbeitet wurde.

DefNotificationProc

```
procedure DefNotificationProc (var Msg: TMessage);
virtual;
```

Führt die Standardverarbeitung für eintreffende Benachrichtigungsbotschaften aus, indem sie das Feld *Result* von *Msg* auf 0 setzt und so anzeigt, daß die Botschaft nicht verarbeitet wurde.

DefScrollProc

```
procedure DefScrollProc(var Msg: TMessage); virtual;
```

Führt die Standardverarbeitung für eintreffende Bildlaufbotschaften durch, indem sie das Feld *Result* von *Msg* auf 0 setzt und damit anzeigt, daß die Botschaft nicht verarbeitet wurde.

DefWndProc

```
procedure DefWndProc(var Msg: TMessage); virtual;
```

Diese Standardfensterprozedur führt nichts aus und wird gewöhnlich überschrieben. Sie beruht darauf, daß das Feld *Result* von *Msg* 0 bleibt, um anzuzeigen, daß die Botschaft nicht verarbeitet wurde. *TWindows* überschreibt *DefWndProc*, um die von Windows zur Verfügung gestellten Standardantworten auf Windows-Botschaften aufzurufen.

Querverweis: *TWindow.DefWndProc*

Destroy

```
procedure Destroy; virtual;
```

Diese Methode bewirkt die Entfernung des dem Schnittstellenobjekt zugeordneten Elements, indem sie veranlaßt, daß das Fensterobjekt eine *wm_Destroy*-Botschaft empfängt, und es damit vom Bildschirm löscht. Sie ruft außerdem *EnableAutoCreate* für jedes aktuell erzeugte untergeordnete Fenster auf, womit sichergestellt wird, daß das Schnittstellenobjekt bei erneuter Erzeugung so aussieht wie vor der Entfernung.

Querverweis: *TWindowsObject.WMDestroy, TWindowsObject.EnableAutoCreate*

DisableAutoCreate

```
procedure DisableAutoCreate;
```

Bewirkt, daß das Schnittstellenobjekt als untergeordnetes Fenster nicht zusammen mit seinen übergeordneten Fenstern erzeugt und dargestellt wird. Rufen Sie *DisableAutoCreate* für Pop-Up-Fenster und Steuerelemente auf, die zu einem späteren Zeitpunkt als ihre übergeordneten Fenster erzeugt und dargestellt werden sollen.

Querverweis: *TWindowsObject.EnableAutoCreate*

DisableTransfer

```
procedure DisableTransfer;
```

Schaltet für das Schnittstellenobjekt den Transfermechanismus aus.

DispatchScroll

```
procedure DispatchScroll(var Msg: TMessage); virtual;
```

Wird von *WMHScroll* und *WMVScroll* aufgerufen, um Bildlaufbotschaften an die betreffenden Objekte zu verteilen.

Querverweis: *TWindowsObject.WMHScroll, TWindowsObject.WMVScroll*

EnableAutoCreate

```
procedure EnableAutoCreate;
```

Stellt sicher, daß das Schnittstellenobjekt als untergeordnetes Fenster zusammen mit seinem übergeordneten Fenster erzeugt und dargestellt wird. Dieses Merkmal ist per Voreinstellung für Fenster und Steuerelemente eingeschaltet und für Dialoge ausgeschaltet. Rufen Sie *EnableAutoCreate* auf, wenn ein Dialog zusammen mit seinen übergeordneten Fenstern erzeugt und dargestellt werden soll.

Querverweis: *TWindowsObject.DisableAutoCreate*

EnableKBHandler

```
procedure EnableKBHandler;
```

Stellt Fenstern und nichtmodalen Dialogen eine Tastaturschnittstelle für untergeordnete Steuerelemente zur Verfügung, ähnlich der für modale Dialoge. Damit kann der Benutzer z.B. mit der Tabulatortaste Steuerelemente auswählen.

EnableTransfer

```
procedure EnableTransfer;
```

Schaltet für das Schnittstellenobjekt den Transfermechanismus ein.

FirstThat

```
function FirstThat(Test: Pointer): PWindowsObject;
```

Durchläuft die Liste der untergeordneten Fenster und übergibt in Aufrufen der booleschen Funktion, auf die *Test* zeigt, der Reihe nach alle untergeordneten Fensterobjekte als Argument. *FirstThat* gibt einen Zeiger auf das erste untergeordnete Fensterobjekt zurück, für das die boolesche Funktion *True* zurückliefert (oder **nil**, wenn kein Aufruf *True* zurückgibt) und stoppt dann den Durchlauf. Sie können zum Beispiel eine Methode *GetFirstChecked* schreiben, die *FirstThat* dazu nutzt, das erste Auswahlfeld zu liefern, das markiert ist.

```
function  MyWindow.GetFirstChecked: PWindowsObject;

  function  IsThisOneChecked(ABox: PWindowsObject):
          Boolean;    far ;
  begin
    IsThisOneChecked := ABox^.GetCheck <> 0;
  end ;

begin
  GetFirstChecked := FirstThat(@IsThisOneChecked);
end ;
```

ForEach

```
procedure ForEach(Action: Pointer);
```

Durchläuft die Liste der untergeordneten Fenster und übergibt in Aufrufen der Prozedur, auf die *Action* zeigt, der Reihe nach alle untergeordneten Fenster als Argument. Sie können zum Beispiel eine Methode *CheckAllBoxes* schreiben, die *ForEach* nutzt, um jedes Auswahlfeld in der Liste untergeordneter Fenster zu markieren:

```
procedure  MyWindow.CheckAllBoxes;

  procedure  CheckTheBox(ABox: PWindowsObject);       far ;
    begin
      PCheckBox(ABox)^.Check;
    end ;

  begin
    ForEach(@CheckAllBoxes);
  end;

  procedure  GetChildPtr (  var S: TStream;   var P);
```

Lädt einen Zeiger *P* auf ein untergeordnetes Fenster aus dem Stream *S*. *GetChildPtr* sollte nur innerhalb eines *Load*-Konstruk-

tors benutzt werden, um Zeigerwerte zu lesen, die von einem Aufruf von *PutChildPtr* von einer *Store*-Methode geschrieben wurden.

Querverweis: *TWindowsObject.GetSiblingPtr, TWindowsObject.-PutSiblingPtr, TWindowsObject.PutChildPtr*

GetClassName

```
function GetClassName: PChar; virtual;
```

Gibt 'TurboWindow', die Standardbezeichnung der Fensterklasse zurück.

GetClient

```
function GetClient: PMDIClient; virtual;
```

Gibt **nil** für alle Nicht-MDI-Schnittstellenobjekte zurück, die keine MDI-Client-Fenster haben. *TMDIWindow* überschreibt diese Methode, um sein MDI-Client-Fenster zu liefern.

GetId

```
function GetId: Integer; virtual;
```

Im allgemeinen wird *GetId* dazu verwendet, den Fensterbezeichner zurückzugeben. Per Voreinstellung liefert *TWindows-Object.GetId* lediglich -1 zurück. *GetId* wird von *TWindow* neudefiniert, um die Steuerelement-ID des Objekts zurückzugeben. Alle anderen Schnittstellenobjekte haben keine Steuerelement-ID.

Querverweis: *TWindow.GetId*

GetSiblingPtr

```
procedure GetSiblingPtr(var S: TStream; var P);
```

Lädt einen Zeiger *P* auf ein Nebenfenster aus dem Stream *S*. Ein Nebenfenster ist ein Fenster, das dasselbe übergeordnete Fenster hat wie dieses Fenster. Zum Beispiel ist das *TGroupBox*-Objekt eines *TCheckBox*-Objektes ein Nebenfenster dieses *TCheckBox* in einem Dialog. *GetSiblingPtr* sollte nur innerhalb eines Konstruktors *Load* verwendet werden, um Zeigerwerte zu lesen, die von einem Aufruf von *PutSiblingPtr* von einer *Store*-Methode geschrieben wurden. Der in *P* geladene Wert wird erst gültig, wenn das übergeordnete Fenster des Fensters seine *Load*-Funktion abgeschlossen hat. Deshalb liefert das Dereferenzieren eines Zeigers auf ein Nebenfenster innerhalb eines *Load*-Konstruktors nicht das richtige Ergebnis.

Querverweis: *TWindowsObject.PutSiblingPtr, TWindowsObject.-GetChildPtr, TWindowsObject.PutChildPtr*

GetWindow-Class

```
procedure GetWindowClass(var AWndClass: TWndClass);
virtual;
```

Dient als Platzhalter für abgeleitete Typen, um den Fensterklassen-Record zu definieren und ihn in *AWndClass* zurückzugeben. Dieses *GetWindowClass* tut nichts.

IsFlagSet

```
function IsFlagSet(Mask: Byte): Boolean;
```

Gibt den Status des Bit-Flags im Feld *Flags*, das durch den Wert in *Mask* bezeichnet ist, zurück. *IsFlagSet* gibt *True* zurück, wenn das Bit-Flag gesetzt ist und *False*, wenn es nicht gesetzt ist.

Querverweis: *TWindowsObject.SetFlags*

Next

```
function Next: PWindowsObject;
```

Gibt einen Zeiger auf das nächste Objekt in der Liste der untergeordneten Fenster des übergeordneten Fensters zurück.

Querverweis: *TWindowsObject.Previous*

Previous

```
function Previous: PWindowsObject;
```

Gibt einen Zeiger auf das vorangegangene Objekt in der Liste der untergeordneten Fenster des übergeordneten Fensters zurück.

Querverweis: *TWindowsObject.Next*

PutChildPtr

```
procedure PutChildPtr(var S: TStream; P: PWindowsObject)
```

Speichert *P*, den Zeiger auf ein untergeordnetes Fenster, im Stream *S*. *PutChildPtr* sollte nur innerhalb einer *Store*-Methode benutzt werden, um Zeigerwerte zu schreiben, die später mit einem Aufruf von *GetChildPtr* von einem *Load*-Konstruktor gelesen werden können.

Querverweis: *TWindowsObject.GetChildPtr, TWindowsObject.-GetSiblingPtr, TWindowsObject.PutSiblingPtr,*

PutSiblingPtr

```
procedure PutSiblingPtr(var S: TStream; P:
PWindowsObject);
```

Speichert *P*, den Zeiger auf ein Nebenfenster, im Stream *S*. Ein Nebenfenster ist ein Fenster mit demselben übergeordneten Fenster wie dieses Fenster. *PutSiblingPtr* sollte nur innerhalb einer *Store*-Methode angewandt werden, um Zeigerwerte zu schreiben, die später mit einem Aufruf an *GetSiblingPtr* von einem *Load*-Konstruktor gelesen werden können.

Querverweis: *TWindowsObject.GetChildPtr, TWindowsObject.-GetSiblingPtr, TWindowsObject.PutChildPtr*

Register

```
function Register: Boolean; virtual;
```

Registriert die Fensterklasse, die in der *GetWindowClass*-Methode des Objekts definiert ist und in der *GetClassName*-Methode bezeichnet ist, wenn sie nicht schon registriert ist. *Register* gibt *True* zurück, wenn die Klasse erfolgreich registriert wurde.

SetFlags

```
procedure SetFlags(Mask: Byte; OnOff: Boolean);
```

Schaltet ein Bit-Flag im *Flags*-Feld – abhängig vom Wert von *OnOff* – an oder aus. Wenn *OnOff True* ist, wird das Bit in *Mask* gesetzt, andernfalls wird es gelöscht. *Mask* kann eine der *wb_*-Konstanten oder eine Kombination von ihnen sein.

Querverweis: *TWindowsObject.IsFlagSet*

SetupWindow

```
procedure SetupWindow; virtual;
```

Führt, normalerweise durch Erzeugung von untergeordneten Fensterelementen – falls solche vorhanden sind -, die Initialisierung des neu erzeugten Schnittstellenelements aus. Sie erzeugt nur diejenigen untergeordneten Fenster, die automatisch erzeugt werden sollen, standardmäßig schließt dies Fenster und Steuerelemente, jedoch keine Dialoge ein.

Ist *Create* nicht in der Lage, ein untergeordnetes Fenster zu erzeugen, setzt sie *Status* auf *em_InvalidChild*. Außerdem ruft *SetupWindow TransferData* auf, um Daten in die neuen untergeordneten Fenster zu kopieren.

Show

```
procedure Show(ShowCmd: Integer);
```

Show stellt das Schnittstellenelement in der Weise auf dem Bildschirm dar, die durch den in *ShowCmd* übergebenen Wert bezeichnet wird. Die zulässigen Werte für *ShowCmd* umfassen:

Wert	Beschreibung
sw_Hide	Ausgeblendet.
sw_Show	Mit der aktuellen Größe und Position des Fensters
sw_ShowMaximized	Maximiert und aktiv.
sw_ShowMinimized	Minimiert und aktiv.
sw_ShowNormal	Wiederhergestellt und aktiv.

Store

```
procedure Store(var S: TStream);
```

Diese Methode speichert das Schnittstellenobjekt im Stream *S*, indem sie das Feld *Status*, andere Attribute und die Größe von *ChildList* schreibt. Anschließend wird jedes untergeordnete Fenster gespeichert.

Transfer

```
function Transfer(DataPtr: Pointer; TransferFlag:
Word): Word; virtual;
```

Gibt 0 zurück. Diese Methode wird von den Nachkommen von *TControl* neudefiniert, um ihre Statusdaten zu und vom Transferpuffer zu transferieren.

Der Rückgabewert von *Transfer* ist die Anzahl der transferierten Datenbytes.

TransferData `procedure TransferData(Direction: Word); virtual;`

Wenn der Transfermechanismus durch Setzen von *TransferBuffer* zu einem Transfer-Record eingeschaltet ist, transferiert diese Methode Daten zwischen dem Puffer und den beteiligten untergeordneten Fenstern.

TransferData ruft die *Transfer-* Methode jedes beteiligten untergeordneten Fensters auf und übergibt einen Zeiger auf den Transferpuffer und die in *Direction* vorgegebene Richtung. *Direction* kann *tf_SetData* oder *tf_GetData* sein.

Querverweis: *TWindowsObject.EnableTransfer, TWindowsObject.DisableTransfer, TWindowsObject.SetupWindow*

WMActivate `procedure WMActivate(var Msg: TMessage); virtual wm_First + wm_Activate;`

Wenn das Schnittstellenobjekt in den Mechanismus der Tastaturverarbeitung einbezogen ist, reagiert diese Methode darauf, daß das Schnittstellenobjekt zum aktiven Fenster wird, mit einem Aufruf der *SetKBHandler*-Methode des Anwendungsobjekts.

Querverweis: *TApplication.SetKBHandler*

WMClose `procedure WMClose(var Msg: TMessage); virtual wm_First + wm_Close;`

Reagiert auf die Aufforderung, das Fenster zu schließen, mit einem Aufruf der *CanClose*-Methode dieses Objekts bzw. der *CanClose*-Methode des Anwendungsobjekts, wenn dieses Objekt das Hauptfenster der Anwendung ist. Wenn *CanClose True* zurückgibt, wird dieses Schnittstellenelement durch Aufruf von *Destroy* freigegeben und entfernt.

Querverweis: *TWindowsObject.Destroy*

WMCommand `procedure WMCommand(var Msg: TMessage); virtual wm_First + wm_Command;`

Stellt den Mechanismus zur Bearbeitung befehlsbasierender und untergeordneter-ID-basierender Botschaften und zum Aufruf der geeigneten Beantwortungsmethoden zur Verfügung.

WMDestroy `procedure WMDestroy(var Msg: TMessage); virtual wm_First + wm_Destroy;`

Für den Fall, daß dieses Objekt das Hauptfenster der Anwendung ist, informiert *WMDestroy* bei Entfernung dieses Schnittstellenelements Windows darüber, daß die Anwendung beendet wird. Dies erzeugt eine *wm_Quit*-Botschaft. Wenn das Objekt nicht das Hauptfenster der Anwendung ist, wird das Standardverhalten des Fensters aufgerufen.

WMHScroll
```
procedure WMHScroll(var Msg: TMessage); virtual
wm_First + wm_HScroll;
```
Fängt horizontale Fensterbildlaufbotschaften ab und ruft *DispatchScroll* auf.

Querverweis: *TWindowsObject.DispatchScroll*

WMNCDestroy
```
procedure WMNCDestroy(var Msg: TMessage); virtual
wm_First + wm_NCDestroy;
```
Reagiert durch Setzen von *HWindow* auf 0 auf die letzte Botschaft, die ein Schnittstellenobjekt empfängt, bevor es entfernt und freigegeben wird.

WMVScroll
```
procedure WMVScroll(var Msg: TMessage); virtual
wm_First + wm_VScroll;
```
Fängt vertikale Fensterbotschaften ab und ruft *DispatchScroll* auf.

Querverweis: *TWindowsObject.DispatchScroll*

6. Außerhalb der Objekte

Dieses Kapitel beschreibt alle Elemente von ObjectWindows, die *nicht* Teil der Standard-Objekthierarchie von ObjectWindows sind. Die Standardobjekte sind alle in Kapitel 5 »Verzeichnis der Objekte« beschrieben.

Die Elemente, die in diesem Kapitel aufgelistet sind, schließen Typen, Konstanten, Variablen, Prozeduren und Funktionen ein, die in den ObjectWindows-Units definiert sind. Eine typische Darstellung sieht so aus:

Sample-Prozedur — Unit von Sample

Deklaration	`procedure Sample(AParameter);`
Beschreibung	*Sample* führt mit ihrem Parameter, *AParameter,* eine nützliche Funktion aus.
Querverweis	Funktion *Example*

Abstract-Prozedur — WObjects

Deklaration	`procedure Abstract;`
Beschreibung	Ein Aufruf dieser Prozedur beendet das Programm mit einem Laufzeitfehler 211. Wird ein *abstrakter Objekttyp* implementiert, sollte man Aufrufe von *Abstract* in denjenigen virtuellen Methoden einsetzen, die von Nachkommen des Typs neu definiert werden müssen. Dies stellt sicher, daß jeder Versuch fehlschlägt, den abstrakten Objekttyp zu instantiieren.
Querverweis	Kapitel 7 »Überblick über ObjectWindows« des *Windows-Programmierhandbuchs*.

AllocMultiSel-Funktion — WObjects

Deklaration	`funktion AllocMultiSel(Size: Integer): PMultiSelRec;`
Beschreibung	Legt einen *TMultiSelRec*-Record mit dem Zählwert *Size* an und reserviert genügend Speicherplatz im Feld *Selections,* um *Size* Auswahlen zu enthalten (0..*Size*-1). Gibt **nil** zurück, wenn

nicht genügend Speicherplatz reserviert werden kann, um den gesamten Record zu speichern.

Querverweis *FreeMultiSel, TMultiSelRec*

Application-Variable **WObjects**

Deklaration
```
Application: PApplication = nil;
```

Beschreibung Die Variable *Application* wird anfangs von *TApplication.Init* auf *@Self* gesetzt und von *TApplication.Done* zu **nil** gelöscht.

Somit zeigt *Application* während der gesamten Ausführung eines ObjectWindows-Programms auf das Anwendungsobjekt.

Querverweis *TApplication.Init*

bf_XXXX-Konstanten **WObjects**

Beschreibung Schalter-, Auswahlfeld- und Schaltfeldobjekte verwenden *bf_*-Konstanten, um ihre drei möglichen Zustände zu definieren.

Die folgenden Schalterflagkonstanten sind definiert:

Konstante	Wert	Bedeutung
bf_Unchecked	0	Element ist nicht ausgewählt.
bf_Checked	1	Element ist ausgewählt.
bf_Grayed	2	Element ist grau hinterlegt.

cm_XXXX-Konstanten **WObjects**

Beschreibung ObjectWindows definiert mehrere Konstanten, die die Bereiche von Befehlsbotschaftskonstanten definieren.

Die folgenden Befehlskonstanten sind definiert:

Konstante	Wert	Bedeutung
cm_first	$A000	Beginn der Befehlsbotschaften
cm_Count	$6000	Anzahl der Befehlsbotschaften
cm_Internal	$FF00	Beginn der für interne Verwendung reservierten Befehlsbotschaften
cm_Reserved		*cm_Internal-cm_First*

*cm_*-Konstanten sind für drei Standardmenüs definiert: File, Edit und Window:

Konstante	Wert	Entsprechung im Menü
cm_EditCut	*cm_Reserved*	Edit|Cut
cm_EditCopy	*cm_Reserved+1*	Edit|Copy
cm_EditPaste	*cm_Reserved+2*	Edit|Paste

Konstante	Wert	Entsprechung im Menü
cm_EditDelete	*cm_Reserved+3*	Edit|Delete
cm_EditClear	*cm_Reserved+4*	Edit|Clear
cm_EditUndo	*cm_IInternalOfset+5*	Edit|Undo
cm_FileNew	*cm_Reserved+6*	File|New
cm_FileOpen	*cm_Reserved+7*	File|Open
cm_MDIFileNew	*cm_Reserved+8*	File|New
cm_MDIFileOpen	*cm_Reserved+9*	File|Open
cm_FileSave	*cm_Reserved+10*	File|Save
cm_FileSaveAs	*cm_Reserved+11*	File|Save As
cm_ArrangeIcons	*cm_Reserved+12*	Window|Arrange Icons
cm_TileChildren	*cm_Reserved+13*	Window|Tile
cm_CascadeChildren	*cm_Reserved+14*	Window|Cascade
cm_CloseChildren	*cm_Reserved+15*	Window|Close All

coXXXX-Konstanten **WObjects**

Beschreibung Die *coXXXX*-Konstanten werden als Parameter *Code* an die Methode *TCollection.Error* übergeben, wenn *TCollection* während einer Operation einen Fehler entdeckt

Die folgenden Standardfehlercodes sind für alle Object-Windows-Kollektionen definiert:

Fehlercode	Wert	Bedeutung
coIndexError	-1	Bereichsüberschreitung des Indizes. Der an die Methode *Error* übergebene Parameter *Info* enthält den ungültigen Index.
coOverflow	-2	Überlauf bei der Kollektion. *TCollection.SetLimit* konnte die Kollektion nicht auf die geforderte Größe ausdehnen. Der an die Methode *Error* übergebene Parameter *Info* enthält die angeforderte Größe.

Querverweis Objekt *TCollection*

em-XXXX-Konstanten **WObjects**

Beschreibung Von ObjectWindows werden mehrere Standardfehlerbedingungen durch Konstanten signalisiert, die mit *em_* beginnen.

Die folgenden Fehler-Flags sind definiert:

Konstante	Wert	Bedeutung
em_OutOfMemory	-1	Eine Speicherreservierung würde den Sicherheitsbereich berühren.
em_InvalidClient	-2	MDI-Client-Fenster konnte nicht erzeugt werden.
em_InvalidChild	-3	Eines oder mehrere der untergeordneten Fenster eines Fensters ist nicht gültig.
em_InvalidWindow	-4	Fenster ist ungültig, weil *Create* keinen Erfolg hatte.
em_InvalidMainWindow	-5	Hauptfenster konnte nicht erzeugt werden.

EmsCurHandle-Variable **WObjects**

Deklaration
```
EmsCurHandle: Word = $FFFF;
```

Beschreibung Enthält das aktuelle EMS-Handle, so wie es von einem *TEmsStream* in die physikalische EMS-Seite 0 abgebildet wurde. *TEmsStream* vermeidet aufwendige Aufrufe zur EMS-Neuabbildung durch Zwischenspeichern des EMS-Status.

Wenn Ihr Programm EMS zu anderen Zwecken einsetzt, sollten Sie sicherstellen, daß *EmsCurHandle* und *EmsCurPage* auf $FFFF gesetzt sind, bevor ein *TEmsStream* verwendet wird – das zwingt den *TEmsStream,* seine Abbildung wiederherzustellen.

Querverweis *TEmsStream.Handle*

EmsCurPage-Variable **WObjects**

Deklaration
```
EmsCurPage: Word= $FFFF;
```

Beschreibung Enthält die aktuelle virtuelle EMS-Seitennummer, wie sie von einem *TEmsStream* in die physikalische EMS-Seite 0 abgebildet wurde. *TEmsStream* vermeidet aufwendige Aufrufe zur EMS-Neuabbildung durch Zwischenspeichern des EMS-Status.

Wenn Ihr Programm EMS für andere Zwecke einsetzt, dann sollten Sie sicherstellen, daß *EmsCurHandle* und *EmsCurPage* auf $FFFF gesetzt sind, bevor ein *TEmsStream* verwendet wird – das zwingt *TEmsStream* dazu, seine Abbildung wiederherzustellen.

Querverweis *TEmsStream.Page*

FreeMultiSel-Prozedur **WObjects**

Deklaration
```
procedure FreeMultiSel(P: PMultiSelRec);
```

Beschreibung Gibt einen von *AllocMultiSel* reservierten*TMultiSelRec*-Record frei.

Querverweis *AllocMultiSel, TMultiSelRec*

id-XXXX-Konstanten **WObjects**

Beschreibung ObjectWindows definiert mehrere Konstanten, die die Bereiche von Botschaften definieren, die auf der ID untergeordneter Fenstern basieren. Die folgenden Botschaftskonstanten sind definiert:

Konstante	Wert	Bedeutung
id_First	$8000	Basis der IDs untergeordneter Fenster
id_Count	$1000	Anzahl der IDs untergeordneter Fenster
id_Internal	*$8F00*	Für internen Gebrauch reserviert
id_Reserved	*id_Internal - id_First;*	
id_First-MDIChild	*id_Reserved*+1	Basis der IDs untergeordneter *MDI-Fenster*
id_MDIClient	*id_Reserved*+2	ID des MDI-Client-Fensters

LongDiv-Funktion **WObjects**

Deklaration
```
function LongDiv(X: LongInt; Y: Integer): Integer;
inline($59/$58/$5A/$F7/$F9);
```

Beschreibung Eine schnelle, durch Inline-Assemblierung codierte Divisionsroutine, die den Integerwert X/Y zurückgibt.

LongMul-Funktion **WObjects**

Deklaration
```
function LongMul(X,Y: Integer): LongInt;
inline($5A/$58/$F7/$EA);
```

Beschreibung Eine schnelle, durch Inline-Assemblierung codierte Multiplikationsroutine, die den LongInt-Wert X*Y zurückgibt.

LongRec-Typ **WObjects**

Deklaration
```
LongRec = record
  Lo,Hi: Word
end;
```

Beschreibung Ein nützlicher Record-Typ zur Bearbeitung von Variablen doppelter Word-Länge.

LowMemory-Funktion **WObjects**

Deklaration
```
function LowMemory: Boolean;
```

Beschreibung Die Funktion *LowMemory* gibt *True* zurück, wenn eine Speicherreservierung in den Sicherheitsbereich am Ende des Heaps übergegriffen hat. Die Größe des Sicherheitsbereiches wird von der Variablen *SafetyPoolSize* festgelegt. *LowMemory* wird automatisch von *TApplication.MakeWindow* und *TApplication.ExecDialog* überprüft, die zum Erzeugen von Fensterelementen verwendet werden sollten. Großverbraucher von Speicher (wie etwa große, komplexe Dialogfenster) sollten von sich aus regelmäßig sicherstellen, daß sie den verfügbaren Speicherplatz nicht überschreiten.

Weitere Einzelheiten über den Gebrauch des Sicherheitsbereiches finden Sie im Abschnitt »Sichere Programme« in Kapitel 19 »Standardrichtlinien für Anwendungen« des *Windows-Programmierhandbuchs.*

Querverweis *MemAlloc, SafetyPoolSize, TApplication.ValidWindow*

MaxCollectionSize-Variable **WObjects**

Deklaration
```
MaxCollectionSize = 65520 div SizeOf(Pointer);
```

Beschreibung *MaxCollectionSize* bestimmt die Maximalzahl von Elementen, die in einer Kollektion enthalten sein können. Dies ist im wesentlichen die Anzahl der Zeiger, die in ein 64KByte-Speichersegment passen.

nf_XXXX-Konstanten **WObjects**

Beschreibung ObjectWindows definiert mehrere Konstanten, die die Bereiche von Benachrichtigungsbotschaften definieren.

Es sind folgende Konstanten definiert:

Konstante	Wert	Bedeutung
nf_First	$9000	Beginn der Benachrichtigungsbotschaften
nf_Count	$1000	Anzahl der Benachrichtigungsbotschaften
nf_Internal	$9F00	Beginn der für internen Gebrauch reservierten Benachrichtigungsbotschaften

PString-Typ **WObjects**

Deklaration
```
PString = ^String;
```

Beschreibung Definiert einen Zeiger auf einen Pascal-String.

PtrRec-Typ **WObjects**

Dklaration
```
PtrRec = record
  Ofs, Seg: Word
  end ;
```

Beschreibung Ein Record, der die Offset- und Segment-Werte eines Zeigers enthält.

RegisterType-Prozedur **WObjects**

Deklaration
```
procedure RegisterType(var S: TStreamRec);
```

Beschreibung Ein ObjectWindows-Objekttyp muß mit dieser Methode registriert werden, bevor er bei der Stream-Ein/Ausgabe benutzt werden kann. Die Standardobjekttypen sind mit *ObjType*-Werten im reservierten Bereich von 0..99 vorregistriert. *RegisterType* erzeugt einen Eintrag in einer verknüpften Liste von *TStreamRec*-Records.

Querverweis *TStream.Get,TStream.Put, TStreamRec*

SafetyPoolSize-Variable **WObjects**

Deklaration
```
SafetyPoolSize: Word = 8192;
```

Beschreibung Definiert die Größe des Speichersicherheitsbereiches. Der Sicherheitsbereich ist ein Puffer am oberen Ende des Heaps, der dazu benutzt wird, sicherzustellen, daß eine Speicherreservierung nicht fehlschlägt. Die Benutzung des Sicherheitsbereiches wird in Kapitel 19 »Standardrichtlinien für Anwendungen« des *Windows-Programmierhandbuch* beschrieben.

Querverweis *MemAlloc, LowMemory, TApplication.ValidWindow*

StrDispose-Prozedur **WObjects**

Deklaration
```
procedure StrDispose(Str: PChar);
```

Beschreibung Gibt einen String frei, der von der Funktion *StrNew* auf dem Heap abgelegt wurde.

Querverweis *StrNew*

StrNew-Funktion **WObjects**

Deklaration
```
function StrNew(Str: PChar): PChar;
```

Beschreibung Dynamische String-Routine. Wenn *S* 0 ist, gibt *StrNew* einen **nil**-Zeiger zurück. Andernfalls werden *Length(S)*+1 Byte reserviert, die eine Kopie von *S* und einen Zeiger auf das erste zurückgegebene Byte enthalten.

Ein mit *StrNew* erzeugter String sollte mit *StrDispose* freigegeben werden.

Querverweis *StrDispose*

stXXXX-Konstanten **WObjects**

Beschreibung Es gibt zwei Arten von Konstanten, die mit »st« beginnen und vom Stream-System von ObjectWindows verwendet werden.

Die folgenden Moduskonstanten werden von *TDosStream* und *TBufStream* verwendet, um den Zugriffsmodus für eine Datei festzulegen, die für einen ObjectWindows-Stream geöffnet ist:

Konstante	Wert	Bedeutung
stCreate	$3C00	Legt eine neue Datei an.
stOpenRead	$3D00	Öffnet eine existierende Datei nur für Lesezugriff.
stOpenWrite	$3D01	Öffnet eine existierende Datei nur für Schreibzugriff.
stOpen	$3D02	Öffnet eine existierende Datei für Lese- und Schreibzugriff.

Die folgenden Werte werden von *TStream.Error* im Feld *TStream.ErrorInfo* zurückgegeben, wenn ein Stream-Fehler auftritt:

Fehlercode	Wert	Bedeutung
stOk	0	Kein Fehler
stError	-1	Zugriffsfehler
stInitError	-2	Stream kann nicht initialisiert werden
stReadError	-3	Leseversuch über das Ende des Streams hinaus
stWriteError	-4	Stream kann nicht erweitert werden
stGetError	-5	*Get* mit unregistriertem Objekttyp
stPutError	-6	*Put* mit unregistriertem Objekttyp

Querverweis *TStream*

StreamError-Variable

WObjects

Deklaration
```
StreamError: Pointer = nil;
```

Beschreibung Wenn *StreamError* nicht **nil** ist, zeigt sie auf eine Prozedur, die von der Fehlermethode eines Streams aufgerufen wird, wenn ein Stream-Fehler auftritt. Die Prozedur muß eine **far**-Prozedur mit einem **var**-Parameter sein, der ein *TStream* ist. Das bedeutet, die Prozedur muß folgendermaßen deklariert werden:

```
procedure MyStreamErrorProc( var S:TStream); far;
```

StreamError erlaubt es Ihnen, die gesamte Stream-Fehlerbehandlung global zu überschreiben. Um die Fehlerbehandlung für einen bestimmten Streamtyp zu ändern, sollten Sie die Methode *Error* dieses Streamtyps überschreiben.

TByteArray-Typ

WObjects

Deklaration
```
TByteArray = array[0..32767] of Byte;
```

Beschreibung Ein ByteArray-Typ zur allgemeinen Verwendung bei der Typumwandlung.

TDialogAttr-Typ

WObjects

Deklaration
```
TDialogAttr = record
  Name: PChar;
  Param: LongInt;
end;
```

Beschreibung *TDialog*-Objekte speichern ihre Attributwerte in einem Record des Typs *TDialogAttr*.

Querverweis *TDialog.Attr*

tf_XXXX-Konstanten

WObjects

Beschreibung Die Methode *Transfer* benutzt Flag-Konstanten, die mit *tf_* beginnen.

Die folgenden Konstanten sind definiert:

Konstanten	Wert	Bedeutung
tf_SizeData	0	Findet die Menge der vom Objekt transferierten Daten heraus.
tf_GetData	1	Ruft Daten aus dem Objekt ab.
tf_SetData	2	Sendet Daten, um den Wert des Objekts zu setzen.

TItemList-Typ **WObjects**

Deklaration
```
TItemList = array[0..MaxCollectionSize - 1] of Pointer;
```

Beschreibung Ein Array von generischen Zeigern, die intern von *TCollection*-Objekten verwendet werden.

TMessage-Typ **WObjects**

Deklaration
```
TMessage = record
  Message: Word;
  case  Integer   of
    0:  (WParam: Word;
         LParam: LongInt;
         Result: LongInt);
    1:  (WParamLo: Byte;
         WParamHi: Byte;
         LParamLo: Word;
         LParamHi: Word;
         ResultLo: Word;
         ResultHi: Word);
end;
```

Beschreibung Die Botschaftsverarbeitungsschleife in *TApplication* verpackt Windows-Botschaftsinformation in *TMessage*-Records, bevor die Information an die passende Botschaftsbeantwortungsmethode übergeben wird.

Querverweis *TApplication.MessageLoop*

TMultiSelRec-Typ **WObjects**

Deklaration
```
TMultiSelRec = record
  Count: Integer;
  Selections:    array [0..32760]   of Integer;
end;
```

Beschreibung *TMultiSelRec*-Records enthalten Listen ausgewählter Optionen zum Transferieren in oder aus einem Listenfenster mit Mehrfachauswahl.

Count kennzeichnet die Anzahl ausgewählter Elemente und *Selections* ist ein Array von Integern mit offenem Ende. Mittels *AllocMultiSel* können Sie einen Record mit genügend viel Auswahloptionen reservieren, der sich daran anpaßt, wieviele ausgewählte Optionen das Listenfenster hat.

Querverweis *AllocMultiSel, FreeMultiSel*

TStreamRec-Typ WObjects

Deklartion

```
TStreamRec = record
  ObjType: Word;
  VmtLink: Word;
  Load: Pointer;
  Store: Pointer;
  Next: Word;
end;
```

Beschreibung

Bevor die Objekte eines ObjectWindows-Objekttyps von einem *TStream*-Objekt geladen oder dort gespeichert werden können, muß dieser einen registrierten *TStreamRec* haben. Die Routine *RegisterType* registriert einen Objekttypen durch Einrichten eines *TStreamRec*-Records. Die Felder im Stream-Registrierungs-Record sind wie folgt definiert:

Feld	Inhalt
ObjType	Eine eindeutige numerische ID für den Objekttypen.
VmtLink	Eine Verbindung zum Eintrag in der VMT des Objekttyps.
Load	Ein Zeiger auf den Konstruktor *Load* des Objekttyps.
Store	Ein Zeiger auf die Methode *Store* des Objekttyps.
Next	Ein Zeiger auf den nächsten *TStreamRec*.

ObjectWindows reserviert für die Objekttypen-IDs (*ObjType*) Werte von 0 bis 999 zum eigenen Gebrauch. Programmierer können ihre Werte im Bereich von 1000 bis 65535 definieren. Vereinbarungsgemäß wird ein *TStreamRec*-Record für einen Objekttypen *Txxxx* mit *Rxxxx* benannt. So wird zum Beispiel ein *TStreamRec* für einen Typ *TCalculator RCalculator* genannt, wie im folgenden Programmtext dargestellt:

```
type
  TCalculator =    object (TDialog)
     constructor   Load( var S: TStream);
     procedure  Store( var S: TStream);
    ...
  end;

const
  RCalculator: TStreamRec = (
  ObjType: 2099;
  VmtLink: Ofs(TypeOf(TCalculator)^);
  Load: @TCalculator.Load;
  Store: @TCalculator.Store);

begin
  RegisterType(RCalculator);
  ...
end;
```

Querverweis *RegisterType*

TWindowAttr-Typ **WObjects**

Deklaration
```
TWindowAttr = record
  Title: PChar;
  Style: LongInt;
  ExStyle: LongInt;
  X,Y,W,H: Integer;
  Param: Pointer;
   case  Integer   of
    0: (Menu: HMenu);  {Handle des Fenstermenis oder...}
    1: (Id: Integer);  {Bezeichner des Steuerelements}
end;
```

Beschreibung *TWindows*-Objekte definieren ihre Attribute in *TWindowAttr*-Records.

Querverweis *TWindow.Attr*

TWordArray-Typ **WObjects**

Deklaration
```
TWordArray = array[0..16383] of Word;
```

Beschreibung Ein Word-Array-Typ für allgemeine Zwecke.

wb_XXXX-Konstanten **WObjects**

Beschreibung Das Feld *Flags* in *TWindowsObject* ist ein Bit-Feld. Auf die Bits kann man mit den Konstanten zugreifen, die mit *wb_* beginnen.

Die folgenden Werte sind definiert:

Konstante	Wert	Bedeutung, wenn gesetzt
wb_KBHandler	$01	Fenster verarbeitet Tastaturereignisse wie ein Dialogfenster.
wb_FromResource	$02	Dialog wurde aus einer Ressource geladen.
wb_AutoCreate	$04	Fenster wird erzeugt, wenn das übergeordnete Fenster erzeugt wird.
wb_MDIChild	$08	Fenster ist ein untergeordnetes MDI-Fenster.
wb_Transfer	$10	Fenster nimmt am *Transfer*-Mechanismus Teil. Standardmäßig wird dieses Bit von *InitResource* gesetzt, von *Init* gelöscht.

Querverweis *TWindowsObject.Flags*

wm_XXXX-Konstanten

WObjects

Beschreibung ObjectWindows definiert einige Konstanten, die sich auf Standard-Windows-Botschaften beziehen und den Bereich der für Windows reservierten Botschaften definieren. Folgende Werte sind definiert:

Konstante	Wert	Bedeutung
wm_First	$0000	Beginn der Windows-Botschaften
wm_Count	$8000	Anzahl der Windows-Botschaften

Querverweis Kapitel 3 »Windows-Botschaften«.

WordRec-Typ

WObjects

Deklaration
```
WordRec = record
  Lo,Hi: Byte
end;
```

Beschreibung Ein Hilfs-Record, der den Zugriff auf das nieder- und das höherwertige Byte eines Word ermöglicht.

Querverweis *LongRec*

Index

A
Abbildungsmodus
  aktueller 218
  festlegen 379
Abmessungen
  Grafikfenster 253, 399
Abstract-Prozedur 653
AccessResource-Funktion 79
AddAtom-Funktion 80
AddFontResource-Funktion 81
AdjustWindowRect-Funktion 81
AdjustWindowRectEx-Funktion 82
Aktivieren
  untergeordnetes Fenster 479
Aktualisierungsregion 251
AllocDStoCSAlias-Funktion 82
AllocMultiSel-Funktion 653
AllocResource-Funktion 83
AllocSelector-Funktion 83
Alte ShowWindow-Konstanten 60
AnimatePalette-Funktion 84
AnsiLower-Funktion 85
AnsiLowerBuff-Funktion 85
AnsiNext-Funktion 85
AnsiPrev-Funktion 86
AnsiToOem-Funktion 86
AnsiToOemBuff-Funktion 86
AnsiUpper-Funktion 87
AnsiUpperBuff-Funktion 87
Anwendungen
  ausführen 299, 441
  bedingtes Schließen 579
  beenden 490
  Botschaftsverarbeitung 581
  Destruktor 579
  Fehler in 579
  Fenster aktivieren in 476
  globale Variable 654
  initialisieren 580
  Konstruktor 579
  Tastatur-Handles 582
AnyPopup-Funktion 88
AppendMenu-Funktion 88
Application-Variable 579, 654
Arc-Funktion 89
ArrangeIconicWindows-Funktion 90
Atome 549
  auffinden 179, 264
  globale 260
  Handles 187
  hinzufügen 80
  löschen 142, 262
  Name 266
  Tabelle 278
Attribute
  Dialogfenster 597
  Fenster 637
  untergeordneter Fenster 614
Ausführung übergeben 448
Ausführungsumgebung
  wiederherstellen 427
Ausrichtung von Text- 397
Auswahlfelder 585ff.
  Auswahlmarkierung 98f., 218
  Benachrichtigungsbotschaften 586
  Gruppenfelder und 586
  Konstruktor 586
  markieren 587
  Markierung aufheben 588
  Objekte verbinden mit 586
  Ressourcen und 586
  Status 449f., 587
  Streams und 586
Automatisches Rollen 626

B
Basiseinheiten
  Dialogfenster 206

Befehle
  Accelerator-Tasten und 480
  Menü- 480
BeginDeferWindowPos-Funktion 90
BeginPaint-Funktion 91
Bereich
  Bildlaufleisten- 237, 389
Bezeichnungen
  Klassen 574
bf_XXXX-Konstanten 654
bi_Xxxx-Konstanten 5
Bibliotheken
  freigeben 184
  Funktionsadressen von 232
  laden 297
Bildlaufleisten 620ff.
  ausblenden 417
  Ausrichtung 620ff.
  Bereich 622
  darstellen 417
  Fenster rollen 626
  Klassenbezeichnung 622
  Konstruktor 621
  Seitengröße 621
  Streams und 621, 624
  Transfer 624
  Zeilengröße 621
Bildlaufsymbol
  Position 623
Bildschirmkontext
  Handles 545
  rollen 356
BitBlt-Funktion 92
Bitmaps 549, 551
  Bits 187, 362, 373
  Datei 551
  erzeugen 105ff.
  gerätespezifische 115
  geräteunabhängige 207
  Größe 188, 362
  Handles 545
  Information 552
  kompatible 108
  komprimieren 420
  kopieren 92
  laden 294
  löschen 144
  Menüoptionen 381
  Pinsel erzeugen aus 116
  strecken 420
  verschieben 420
  verwerfbare 117
  vordefinierte 48
bm_GetCheck-Botschaft 449
bm_GetState-Botschaft 449
bm_SetCheck-Botschaft 450
bm_SetState-Botschaft 450
bm_SetStyle-Botschaft 450
bn_Xxxx-Konstanten 5
Bool-Typ 545
Botschaften 449, 568
  abrufen 221, 336f.
  an Dialogfensterelemente senden 599
  Befehls- 480
  Dialogfenster und 288
  dynamischer Datenaustausch 483
  Parameter 449
  Position 222
  reagieren auf 350
  senden 361
  übergeben 150
  übergebene 279
  übermitteln 341
  übersetzen 431
  warten auf 439
  Zeit 222
Botschaftsfenster 318f.
Botschaftsschlange
  erzeugen 381
BringWindowToTop-Funktion 93
bs_Xxxx-Konstanten 6
BuildCommDCB-Funktion 94
Bytes
  Zeiger 548

C

CallMsgFilter-Funktion 94
CallWindowProc-Funktion 95
Carets
  anzeigen 416
  ausblenden 276
  Blinkrate 189, 365
  entfernen 144
  erzeugen 107
  Position 190
Catch-Funktion 96
cb_AddString-Botschaft 451
cb_DeleteString-Botschaft 451
cb_Dir-Botschaft 452
cb_FindString-Botschaft 452
cb_GetCount-Botschaft 452
cb_GetCurSel-Botschaft 453
cb_GetEditSel-Botschaft 453
cb_GetItemData-Botschaft 453

cb_GetLBText-Botschaft 454
cb_GetLBTextLen-Botschaft 454
cb_InsertString-Botschaft 454
cb_LimitText-Botschaft 455
cb_ResetContent-Botschaft 455
cb_SelectString-Botschaft 455
cb_SetCurSel-Botschaft 456
cb_SetEditSel-Botschaft 456
cb_SetItemData-Botschaft 457
cb_ShowDropDown-Botschaft 457
cb_Xxxx-Konstanten 7
cbm_Init-Konstanten 7
cbn_Xxxx-Konstanten 8
cbs_Xxxx-Konstanten 8f.
cc_Xxxx-Konstanten 10
cchDeviceName-Konstante 12
ce_Xxxx-Konstanten 10
cf_Xxxx-Konstanten 11
ChangeClipboardChain-Funktion 96
ChangeMenu-Funktion 96
ChangeSelector-Funktion 97
CheckDlgButton-Funktion 98
CheckMenuItem-Funktion 98
CheckRadioButton-Funktion 99
ChildWindowFromPoint-Funktion 100
Chord-Funktion 100
ClearCommBreak-Funktion 101
Client-Fenster 553, 618
  Attribute 614
  initialisieren 618
  Klassenbezeichnung 615
  Konstruktor 614
  MDI 616
  nebeneinander anordnen 615
  Streams und 614
  Symbole anordnen 615
  überlappen 615
ClientToScreen-Funktion 101
clip_Xxxx-Konstanten 13
ClipCursor-Funktion 102
Clipping-Region
  definieren 173
  verschieben 327
CloseClipboard-Funktion 102
CloseComm-Funktion 102
CloseMetaFile-Funktion 103
CloseSound-Funktion 103
CloseWindow-Funktion 103
cm_XXXX-Konstanten 654
Code-Segment
  ermitteln 196
color_Xxxx-Konstanten 13
com_Xxxx-Konstanten 14
CombineRgn-Funktion 103
CopyMetaFile-Funktion 104
CopyRect-Funktion 104
CountClipboardFormats-Funktion 105
CountVoiceNotes-Funktion 105
coXXXX-Konstanten 655
cp_Xxxx-Konstanten 14
CreateBitmap-Funktion 105
CreateBitmapIndirect-Funktion 106
CreateBrushIndirect-Funktion 106
CreateCaret-Funktion 107
CreateCompatibleBitmap-Funktion 108
CreateCompatibleDC-Funktion 108
CreateCursor-Funktion 109
CreateDC-Funktion 110
CreateDialog-Funktion 111
CreateDialogIndirect-Funktion 112f.
CreateDialogIndirectParam-Funktion 114
CreateDialogParam-Funktion 115
CreateDIBitmap-Funktion 115
CreateDIBPatternBrush-Funktion 116
CreateDiscardableBitmap-Funktion 117
CreateEllipticRgn-Funktion 118
CreateEllipticRgnIndirect-Funktion 118
CreateFont-Funktion 118f.
CreateFontIndirect-Funktion 121
CreateHatchBrush-Funktion 121
CreateIC-Funktion 122
CreateIcon-Funktion 122
CreateMenu-Funktion 123
CreateMetaFile-Funktion 123
CreatePalette-Funktion 124
CreatePatternBrush-Funktion 124
CreatePen-Funktion 125
CreatePenIndirect-Funktion 125
CreatePolygonRgn-Funktion 126
CreatePolyPolygonRgn-Funktion 126
CreatePopupMenu-Funktion 127
CreateRectRgn-Funktion 128
CreateRectRgnIndirect-Funktion 128
CreateRoundRectRgn-Funktion 128
CreateSolidBrush-Funktion 129
CreateWindow-Funktion 129ff.
CreateWindowEx-Funktion 134
cs_Xxxx-Konstanten 14
ctlcolor_Xxxx-Konstanten 15
cw_UseDefault-Konstante 16

D

Datei-Handles
  verfügbare 378

Dateien
  Dateizeiger 293
  erzeugen 331
  Handles 601
  lesen 312
  löschen 331
  öffnen 291, 311, 331
  schließen 291
  schreiben 315
  temporäre 245f.
Dateinamen
  Listen und 452, 467
  Modul- 224
dc_Xxxx-Konstanten 16
dcb_Xxxx-Konstanten 16
dde_Xxxx-Konstanten 17
DebugBreak-Prozedur 135
Debugger
  Strings senden an 333
DefDlgProc-Funktion 135
DeferWindowPos-Funktion 136f.
DefFrameProc-Funktion 138
DefHookProc-Funktion 139
DefineHandleTable-Funktion 140
DefMDIChildProc-Funktion 140f.
DefWindowProc-Funktion 142
DeleteAtom-Funktion 142
DeleteDC-Funktion 143
DeleteMenu-Funktion 143
DeleteMetaFile-Funktion 144
DeleteObject-Funktion 144
Desktop-Handle 202
DestroyCaret-Prozedur 144
DestroyCursor-Funktion 145
DestroyIcon-Funktion 145
DestroyMenu-Funktion 145
DestroyWindow-Funktion 146
DialogBox-Funktion 146f.
DialogBoxIndirect-Funktion 148
DialogBoxIndirectParam-Funktion 149
DialogBoxParam-Funktion 150
Dialogfenster 600
  abbrechen 598, 600
  Attribute 597
  ausführen 599
  Basiseinheiten 206
  Botschaft 318f.
  Botschaftsbearbeitung 135
  Destruktor 598
  Elemente 599
  erzeugen 598, 600
  Fensterklassen 601
  initialisieren 497, 600
  Konstruktor 597, 600
  löschen und freigeben 598
  modale 162, 580, 596ff.
  nichtmodale 596ff.
  OK-Schalter und 599
  Schalter 457f.
  schließen 598, 601
  Schriften und 493
  Standardbotschaftsverarbeitung 598
  Streams und 598
DIB_Xxxx-Konstanten 17
DispatchMessage-Funktion 150
dlgc_Xxxx-Konstanten 17
DlgDirList-Funktion 151
DlgDirListComboBox-Funktion 152f.
DlgDirSelect-Funktion 154
DlgDirSelectComboBox-Funktion 155
DlgWindowExtra-Konstante 18
dm_GetDefID-Botschaft 457
dm_SetDefID-Botschaft 458
dm_Xxxx-Konstanten 18
dmbin_Xxxx-Konstanten 19
dmcolor_Xxxx-Konstanten 19
dmdup_Xxxx-Konstanten 19
dmorient_Xxxx-Konstanten 19
dmpaper_Xxxx-Konstanten 19
dmres_Xxxx-Konstanten 20
DOS3Call-Prozedur 156
DPtoLP-Funktion 156
DrawFocusRect-Prozedur 157
DrawIcon-Funktion 157
DrawMenuBar-Prozedur 158
DrawText-Funktion 158
drive_Xxxx-Konstanten 20
Drucker-Escape-Codes 21, 23
ds_Xxxx-Konstanten 24
dt_Xxxx-Konstanten 25
Dynamischer Datenaustausch
  Benachrichtigung 483

E

Editierfelder
  formatieren 458
  formatierendes Rechteck 460, 464
  Klassenbezeichnung 605
  Konstruktor 603
  mehrzeilige 459, 462, 602
  rollen 462
  rückgängig machen 458
  Streams und 603, 607
  Tabulatorpositionen setzen 465

Text 494
verändern 460
Zeilen abrufen 459
Zeilenanzahl 460
Zeilenindex 461
Zeilenlänge 462
Einheiten
Dialogfenster 317
Ellipse-Funktion 159
Ellipsen zeichnen 159
em-XXXX-Konstanten 655
em_CanUndo-Botschaft 458
em_EmptyUndoBuffer-Botschaft 458
em_FmtLines-Botschaft 458
em_GetHandle-Botschaft 459
em_GetLine-Botschaft 459
em_GetLineCount-Botschaft 460
em_GetModify-Botschaft 460
em_GetRect-Botschaft 460
em_GetSel-Botschaft 460
em_LimitText-Botschaft 461
em_LineFromChar-Botschaft 461
em_LineIndex-Botschaft 461
em_LineLength-Botschaft 462
em_LineScroll-Botschaft 462
em_ReplaceSel-Botschaft 462
em_SetHandle-Botschaft 463
em_SetModify-Botschaft 463
em_SetPasswordChar-Botschaft 463
em_SetRect-Botschaft 464
em_SetRectNP-Botschaft 464
em_SetSel-Botschaft 464
em_SetTabStops-Botschaft 465
em_SetWordBreak-Botschaft 465
em_Undo-Botschaft 466
EmptyClipboard-Funktion 159
EmsCurHandle-Variable 656
EmsCurPage-Variable 656
en_Xxxx-Konstanten 26
EnableHardwareInput-Funktion 160
EnableMenuItem-Funktion 160
EnableWindow-Funktion 161
EndDeferWindowPos-Prozedur 161
EndDialog-Prozedur 162
EndPaint-Prozedur 162
EnumChildWindows-Funktion 163
EnumClipboardFormats-Funktion 164
EnumFonts-Funktion 164f.
EnumMetaFile-Funktion 166
EnumObjects-Funktion 167
EnumProps-Funktion 168
EnumTaskWindows-Funktion 169
EnumWindows-Funktion 170
EqualRect-Funktion 170
EqualRgn-Funktion 171
Ereignisse
Schwellenwert 251
Timer- 290
überprüfen auf 213
es_Xxxx-Konstanten 27
Escape-Funktion 171
Escape-Konstanten für serielle Schnittstellen 28
EscapeCommFunction-Funktion 172
eto_Xxxx-Konstanten 29
ev_Xxxx-Konstanten 29
ExcludeClipRect-Funktion 173
ExcludeUpdateRgn-Funktion 173
ExitWindows-Funktion 174
ExtFloodFill-Funktion 174
ExtTextOut-Funktion 175

F

Farben 553
aktualisieren 436
annähern 225
auswählen 482
Bildschirmelemente 239
einstellen 392
Hintergrund 188, 363
palettenindizierte 554
RGB 553, 572
Standardsystem- 483
Text 249, 396, 482
FatalAppExit-Prozedur 177
FatalExit-Prozedur 177
Fehler
Kollektionen 590
Streams 633, 661
Fehlerbearbeitung
festlegen 377
Fenster 637
Abmessungen 255f.
aktives 185
aktivieren 161, 640
aktualisieren 436
an oberste Stelle bringen 93
Attribute 637
auflisten 169f.
ausblenden 418
ausschalten 490
Bildschirmkontext 254
blinkende 181
Botschaften 347, 361

Client-Bereich 193, 438
darstellen 418
Desktop- 202
Destruktor 638
eingeschaltetes 290
einrichten 640
enthaltene Punkte 440
ermitteln 254
erzeugen 129ff., 482, 638, 640
Fokus und 640
fokussierte 185, 500
freigeben und entfernen 146, 488, 639f.
Größe 641
Größe ändern 81f., 408, 493
gültiges 582
Handles 547
IDs 639
initialisieren 482
Klassen 574, 639, 647
Konstruktor 638
Mausklicks 641
Menüs 241, 380
nächstes 226
nebeneinander anordnen 615
Objekten zuordnen 637
Pop-up- 418
Position 408
Ressourcen und 638
rollen 356, 627
schließen 103, 480
sichtbares 290
Standardbotschafts-verarbeitung 142, 637, 639
symbolisches 289, 497
systemmodales 240, 393
Tasknummer 257
Text 415, 494
Textlänge 495
übergeordnete 181, 228, 383
überlappen 615
Überschrift 258, 415
untergeordnete 287
Ursprung 256, 329, 407
vergrößertes 290
vorhandenes 289
zeichnen 639, 641
zusätzliche Bytes 255, 258, 574
Fensterelemente erzeugen 580
Fensterfunktionen aufrufen 95
ff_Xxxx-Konstanten 30
FillRect-Funktion 178
FillRgn-Funktion 179
Filterfunktionen
installieren 409ff.
Filterverwaltungsfunktionen
löschen 433
nächste aufrufen 139
FindAtom-Funktion 179
FindResource-Funktion 179
FindWindow-Funktion 181
FlashWindow-Funktion 181
FloodFill-Flags 30
FloodFill-Funktion 182
FlushComm-Funktion 182
Fokus setzen 378
fokussiertes untergeordnetes Fenster 637
Formate
Zwischenablage 476
FrameRect-Prozedur 183
FrameRgn-Funktion 183
FreeLibrary-Prozedur 184
FreeModule-Funktion 184
FreeMultiSel-Prozedur 657
FreeProcInstance-Prozedur 184
FreeResource-Funktion 185
FreeSelector-Funktion 185
Füllmodus
Polygone 229, 384
fw_Xxxx-Konstanten 30

G

gcl_Xxxx-Konstanten 31
gcw_Xxxx-Konstanten 31
Gerätefähigkeits-Konstanten 31
Gerätekontexte
erzeugen 108, 110
freigeben 348
Grafikfensterabmessungen 253
Grafikfensterursprung 253
Handles 545
kompatible 108
löschen 143
Palettenauswahl 360
speichern 353
wiederherstellen 351
Gerätemodi 561
ändern 489
Gerätesteuerblöcke
erstellen 94
GetActiveWindow-Funktion 185
GetAspectRatioFilter-Funktion 186
GetAsyncKeyState-Funktion 186
GetAtomHandle-Funktion 187

GetAtomName-Funktion 187
GetBitmapBits-Funktion 187
GetBitmapDimension-Funktion 188
GetBkColor-Funktion 188
GetBkMode-Funktion 188
GetBrushOrg-Funktion 188
GetBValue-Funktion 189
GetCapture-Funktion 189
GetCaretBlinkTime-Funktion 189
GetCaretPos-Prozedur 190
GetCharWidth-Funktion 190
GetClassInfo-Funktion 190
GetClassLong-Funktion 191
GetClassName-Funktion 192
GetClassWord-Funktion 192
GetClientRect-Prozedur 193
GetClipboardData-Funktion 193
GetClipboardFormatName-Funktion 194
GetClipboardOwner-Funktion 195
GetClipboardViewer-Funktion 195
GetClipBox-Funktion 195
GetCodeHandle-Funktion 196
GetCodeInfo-Prozedur 196
GetCommError-Funktion 197
GetCommEventMask-Funktion 199
GetCommState-Funktion 199
GetCurrentPDB-Funktion 200
GetCurrentPosition-Funktion 200
GetCurrentTask-Funktion 200
GetCurrentTime-Funktion 200
GetCursorPos-Prozedur 201
GetDC-Funktion 201
GetDCOrg-Funktion 202
GetDesktopWindow-Funktion 202
GetDeviceCaps-Funktion 202ff.
GetDialogBaseUnits-Funktion 206
GetDIBits-Funktion 207
GetDlgCtrlID-Funktion 208
GetDlgItem-Funktion 208
GetDlgItemInt-Funktion 209
GetDlgItemText-Funktion 209
GetDOSEnvironment-Funktion 210
GetDoubleClickTime-Funktion 210
GetDriveType-Funktion 211
GetEnvironment-Funktion 211
GetFocus-Funktion 212
GetFreeSpace-Funktion 212
GetGValue-Funktion 213
GetInputState-Funktion 213
GetInstanceData-Funktion 213
GetKBCodePage-Funktion 214
GetKeyboardState-Prozedur 214
GetKeyboardType-Funktion 215
GetKeyNameText-Funktion 216
GetKeyState-Funktion 217
GetLastActivePopup-Funktion 217
GetMapMode-Funktion 218
GetMenu-Funktion 218
GetMenuCheckMarkDimensions-Funktion 218
GetMenuItemCount-Funktion 218
GetMenuItemID-Funktion 219
GetMenuState-Funktion 219
GetMenuString-Funktion 220
GetMessage-Funktion 221
GetMessagePos-Funktion 222
GetMessageTime-Funktion 222
GetMetaFile-Funktion 223
GetMetaFileBits-Funktion 223
GetModuleFileName-Funktion 224
GetModuleHandle-Funktion 224
GetModuleUsage-Funktion 224
GetNearestColor-Funktion 225
GetNearestPaletteIndex-Funktion 225
GetNextDlgGroupItem-Funktion 225
GetNextDlgTabItem-Funktion 226
GetNextWindow-Funktion 226
GetNumTasks-Funktion 227
GetObject-Funktion 227
GetPaletteEntries-Funktion 228
GetParent-Funktion 228
GetPixel-Funktion 229
GetPolyFillMode-Funktion 229
GetPriorityClipboardFormat-Funktion 229
GetPrivateProfileInt-Funktion 230
GetPrivateProfileString-Funktion 231
GetProcAddress-Funktion 232
GetProfileInt-Funktion 233
GetProfileString-Funktion 233
GetProp-Funktion 235
GetRgnBox-Funktion 235
GetROP2-Funktion 236
GetRValue-Funktion 236
GetScrollPos-Funktion 236
GetScrollRange-Prozedur 237
GetStockObject-Funktion 238
GetStretchBitMode-Funktion 239
GetSubMenu-Funktion 239
GetSysColor-Funktion 239
GetSysModalWindow-Funktion 240
GetSystemDirectory-Prozedur 240
GetSystemMenu-Funktion 241

GetSystemMetrics-Funktion 242
GetSystemPaletteEntries-Funktion 243
GetSystemPaletteUse-Funktion 244
GetTabbedTextExtent-Funktion 244
GetTempDrive-Funktion 245
GetTempFileName-Funktion 246
GetTextAlign-Funktion 247
GetTextCharacterExtra-Funktion 248
GetTextColor-Funktion 249
GetTextExtent-Funktion 249
GetTextFace-Funktion 250
GetTextMetrics-Funktion 250
GetThresholdEvent-Funktion 250
GetThresholdStatus-Funktion 251
GetTickCount-Funktion 251
GetTopWindow-Funktion 251
GetUpdateRect-Funktion 251
GetUpdateRgn-Funktion 252
GetVersion-Funktion 253
GetViewportExt-Funktion 253
GetViewportOrg-Funktion 253
GetWindow-Funktion 254
GetWindowDC-Funktion 254
GetWindowExt-Funktion 255
GetWindowLong-Funktion 255
GetWindowOrg-Funktion 256
GetWindowRect-Prozedur 256
GetWindowsDirectory-Funktion 257
GetWindowTask-Funktion 257
GetWindowText-Funktion 258
GetWindowTextLength-Funktion 258
GetWindowWord-Funktion 258
GetWinFlags-Funktion 259
GlobalAddAtom-Funktion 260
GlobalAlloc-Funktion 260f.
GlobalCompact-Funktion 262
GlobalDeleteAtom-Funktion 262
GlobalDiscard-Funktion 263
GlobalDosAlloc-Funktion 263
GlobalDosFree-Funktion 264
globale Handles 563
globaler Heap
  freier Platz 212
GlobalFindAtom-Funktion 264
GlobalFix-Prozedur 264
GlobalFlags-Funktion 265
GlobalFree-Funktion 266
GlobalGetAtomName-Funktion 266
GlobalHandle-Funktion 266
GlobalLock-Funktion 267
GlobalLRUNewest-Funktion 267
GlobalLRUOldest-Funktion 268
GlobalNotify-Prozedur 268
GlobalPageLock-Funktion 269
GlobalPageUnlock-Funktion 270
GlobalReAlloc-Funktion 270f.
GlobalSize-Funktion 272
GlobalUnfix-Funktion 272
GlobalUnlock-Funktion 273
GlobalUnWire-Funktion 273
GlobalWire-Funktion 274
gmem_Xxxx-Konstanten 33
Grafikfenster
  Abmessungen 253, 399
  Ursprung 328, 400
GrayString-Funktion 274f.
Gruppen
  Auswahl 610
  Benachrichtigung 610
  Klassenbezeichnung 610
  Konstruktor 610
  Objekte zuordnen 610
  Ressourcen und 610
  Streams und 611
Gruppenfelder
  Auswahlfelder und 586
gw_Xxxx-Konstanten 34
gwl_Xxxx-Konstanten 34
gww_Xxxx-Konstanten 35

H

Handle-Tabelle 563
Handles
  Bitmaps 545
  Dialogfensterelemente 599
  DOS-Datei 601
  EMS 656
  Fenster 547, 642
  Gerätekontext- 545
  globale 563
  Mauszeiger- 545
  Menüs 546
  Paletten- 546
  Pinsel- 545
  Regionen 546
  Schriften 546
  Stifte 546
  Strings 546
  Symbole 546
  Zeiger 547f.
Hauptfenster
  erzeugen 580
  initialisieren 580
HBitmap-Typ 545

HBrush-Typ 545
HCursor-Typ 545
HDC-Typ 545
Heap
  globaler 260ff.
  lokaler 305, 308
help_Xxxx-Konstanten 35
HFont-Typ 546
HiByte-Funktion 276
HIcon-Typ 546
HideCaret-Prozedur 276
Hilfesystem
  aufrufen 442f.
HiliteMenuItem-Funktion 277
Hintergrund
  Farbe 188, 363
  löschen 491
  Modus 188, 363
  neu zeichnen 491
Hintergrundmodi-Konstanten 36
HiWord-Funktion 278
HMenu-Typ 546
HPalette-Typ 546
HPen-Typ 546
HRgn-Typ 546
hs_Xxxx-Konstanten 36
HStr-Typ 546
htXxxx-Konstanten 36
Hüllkurve
  Klang-Warteschlange 402
HWnd-Typ 547

I
id-XXXX-Konstanten 657
id_Xxxx-Konstanten 37
idc_Xxxx-Konstanten 38
idi_Xxxx-Konstanten 38
IDs
  Fenster 639
  Menüoptionen 219
  Schnittstellenobjekte 647
ie_Xxxx-Konstanten 38
InflateRect-Prozedur 278
InitAtomTable-Funktion 278
Initialisierungsdateien
  private 230f.
  schreiben in 444f.
InSendMessage-Funktion 279
InsertMenu-Funktion 279, 281
Instanzdaten
  kopieren 213
Integer
  Zeiger 548
IntersectClipRect-Funktion 282
IntersectRect-Funktion 283
InvalidateRect-Prozedur 283
InvalidateRgn-Prozedur 284
InvertRect-Prozedur 285
InvertRgn-Funktion 285
IsCharAlpha-Funktion 286
IsCharAlphaNumeric-Funktion 286
IsCharLower-Funktion 286
IsCharUpper-Funktion 287
IsChild-Funktion 287
IsClipboardFormatAvailable-
  Funktion 287
IsDialogMessage-Funktion 288
IsDlgButtonChecked-Funktion 288
IsIconic-Funktion 289
IsRectEmpty-Funktion 289
IsWindow-Funktion 289
IsWindowEnabled-Funktion 290
IsWindowVisible-Funktion 290
IsZoomed-Funktion 290

K
KillTimer-Funktion 290
Klang-Warteschlange
  abspielen 419
  Größe festlegen 404
  leeren 419
  Schwellenwert 405
  synchronisieren 425
Klangausgabegerät
  öffnen 333
Klänge
  beenden 103
  einstellen 390
  für Klang-Warteschlange 404f.
  Warnton 318
  warten auf Status 440
  Wiedergabe 419
Klassen
  Bezeichnungen 574
  Fenster- 574, 639, 647
  Hintergrundfarben 574
  Information abrufen 190
  Mauszeiger 574
  Menüs 574
  Namen 192
  Registrierung 346, 596
  Registrierung aufheben 435
  Symbole 574
  zusätzliche Bytes 191f.

Kollektionen 588f., 591, 593, 662
  Destruktor 589
  Elemente 588
  Fehler 590, 655
  Größe 588, 593
  Konstanten 655
  Kontstruktor 589
  sortierte 628f.
  Streams und 589, 592f.
  String- 636
Kombinationsfenster 594
  aktuelle Auswahl in 155
  Drop-Down-Listen 457
  Einträge 455, 488
  Klassenbezeichnung 595
  Konstruktor 594
  Listen 595
  Objekte zuordnen 594
  Ressourcen und 594
  Stile 594
  Streams und 595
  Text 453, 455, 494
  Verzeichnislisten in 152f.
Koordinaten
  aktuelle Position 200, 324
  Bildschirm 101, 355
  Client 101, 355
  Client-Bereich 193
  Punkte 571
  Rechtecks- 385

L

Laufwerke
  temporäre Dateien 245
  Typbestimmung 211
Lautsprecher 103
lb_AddString-Botschaft 466
lb_DeleteString-Botschaft 467
lb_Dir-Botschaft 467
lb_FindString-Botschaft 468
lb_GetCount-Botschaft 468
lb_GetCursel-Botschaft 468
lb_GetHorizontalExtent-Botschaft 469
lb_GetItemData-Botschaft 469
lb_GetItemRect-Botschaft 469
lb_GetSel-Botschaft 470
lb_GetSelCount-Botschaft 470
lb_GetSelItemS-Botschaft 470
lb_GetText-Botschaft 470
lb_GetTextLen-Botschaft 471
lb_GetTopIndex-Botschaft 471
lb_InsertString-Botschaft 471
lb_ResetContent-Botschaft 472
lb_SelectStrings-Botschaft 472
lb_SelItemRange-Botschaft 473
lb_SetColumnWidth-Botschaft 473
lb_SetCursel-Botschaft 473
lb_SetHorizontalExtent-Botschaft 474
lb_SetItemData-Botschaft 474
lb_SetSel-Botschaft 474
lb_SetTabStops-Botschaft 475
lb_SetTopIndex-Botschaft 475
lb_Xxxx-Konstanten 39
lbn_Xxxx-Konstanten 39
lbs_Xxxx-Konstanten 39
lc_Xxxx-Konstanten 41
_lclose-Funktion 291
_lcreat-Funktion 291
Leistungsmerkmale
  abrufen 202ff.
  Clipping 14
  Geräte 31
  Kurven 10
  Linie 41
  Raster 54
  Text 67
  Vielecke 51
lf_FaceSize-Konstante 41
LimitEmsPages-Prozedur 292
LineDDA-Prozedur 292
LineTo-Funktion 293
Linien zeichnen 293
Listen 611ff.
  aktuelle Auswahl in 154
  ausgewählte Elemente 612
  benutzerdefinierte Einträge 481
  Dateinamen und 467
  Einträge 472, 612f.
  Klassenbezeichnung 612
  Konstruktor 611
  löschen 612
  Mehrfachauswahl 613, 653, 662
  Spaltenbreite 473
  Strings in 466, 468
  Tabulatoren 475
  Tastendrücke und 478
  Transfer-Records 613
  Umrandungen 469
  Verzeichnislisten in 151
_llseek-Funktion 293
lmem_Xxxx-Konstanten 41
LoadAccelerators-Funktion 294
LoadBitmap-Funktion 294
LoadCursor-Funktion 295

LoadIcon-Funktion 297
LoadLibrary-Funktion 297
LoadMenu-Funktion 298
LoadMenuIndirect-Funktion 299
LoadModule-Funktion 299
LoadResource-Funktion 301
LoadString-Funktion 301
LoByte-Funktion 302
LocalAlloc-Funktion 302
LocalCompact-Funktion 303
LocalDiscard-Funktion 304
LocalFlags-Funktion 304
LocalFree-Funktion 305
LocalHandle-Funktion 305
LocalInit-Funktion 305
LocalLock-Funktion 306
LocalReAlloc-Funktion 306f.
LocalShrink-Funktion 308
LocalSize-Funktion 308
LocalUnlock-Funktion 309
LockData-Funktion 309
LockResource-Funktion 309
LockSegment-Funktion 310
LongDiv-Funktion 657
LongMul-Funktion 657
LongRec-Typ 657
_lopen-Funktion 311
LowMemory-Funktion 658
LoWord-Funktion 312
LPHandle-Typ 547
LPtoDP-Funktion 312
LPTx-Konstante 42
LPVoid-Typ 547
_lread-Funktion 312
lstrcat-Funktion 313
lstrcmp-Funktion 313
lstrcmpi-Funktion 314
lstrcpy-Funktion 314
lstrlen-Funktion 315
_lwrite-Funktion 315

M

ma_Xxxx-Konstanten 42
MakeIntAtomType 547
MakeIntResource-Typ 547
Makelong-Funktion 316
MakeProcInstance-Funktion 316
MapDialogRect-Prozedur 317
MapVirtualKey-Funktion 317
Maße
  System 242
  Text 572f.
Maus
  -Fang 189, 348, 364
  -Klicks 501
  Doppelklick 376, 503
  -Tasten 423, 503
Mausfang aufheben 348
Mauszeiger
  beschränken 102
  Clipping 102
  Form 372
  Handles 545
  Klassen 574
  laden 295
  Position 201, 222, 372
MaxCollectionSize-Variable 658
mb_Xxxx-Konstanten 42f.
MDI 615, 617
  Fenster 616
  Fensterklasse 618
  Streams und 619
  Symbole 617
  untergeordnete Fenster 617
Meldungsfenster 477
Menüleisten
  zeichnen 158
Menüoptionen
  aktivieren 160
  ändern 321, 323
  Anzahl 218
  Bitmaps als 381
  einfügen 279, 281
  einrichten 380
  hervorheben 277
  ID-Nummern 219
  Label 220
  laden 298
  löschen 349
  markieren 98
  MDI-Fenster 616
Menüs 491
  Auswahlmarkierung 218
  entfernen 145
  erzeugen 123
  Handles 218, 546
  initialisieren 497
  Klassen 574
  löschen 143
  Optionen anfügen 88
  Pop-up- 127
MessageBeep-Prozedur 318
MessageBox-Funktion 318f.
meta_Xxxx-Konstanten 44

Metafiles
  abrufen 223
  auflisten 166
  ausführen 339
  Bilder 567
  erzeugen 123
  Größe von 223
  Header 568
  kopieren 104
  löschen 144
  Records 568
  schließen 103
Methoden
  abstrakte 653
mf_Xxxx-Konstanten 45
Millisekunden
  Anzahl seit Systemstart 251
mk_Xxxx-Konstanten 46
mm_Xxxx-Konstanten 47
Modi
  aufheben 477
  Hintergrund- 188, 363
  Streckungs- 239, 390
  Zeichen- 387
ModifyMenu-Funktion 321, 323
Module
  ausführbare Dateien 224
  Bibliotheken 297
  freigeben 184
  Handles 224
  Referenzzähler abfragen 224
MoveTo-Funktion 324
MoveWindow-Prozedur 324
msgf_Xxxx-Konstanten 47
MulDiv-Funktion 325
Muster 571

N
Nebenfenster
  Streams und 648
NetBIOSCall-Prozedur 325
nf_XXXX-Konstanten 658
Noten
  Anzahl in Voice 105

O
obj_Xxxx-Konstanten 47
Objekte
  auflisten 167
  auswählen 358f.
  Basis- 619
  laden 227
  löschen 144
  vordefinierte 238
obm_Xxxx-Konstanten 48
oda_Xxxx-Konstanten 48
ods_Xxxx-Konstanten 48
odt_Xxxx-Konstanten 49
OemKeyScan-Funktion 326
OemToAnsi-Funktion 326
OemToAnsiBuff-Prozedur 327
of_Xxxx-Konstanten 49
OffsetClipRgns-Funktion 327
OffsetRect-Prozedur 327
OffsetRgn-Funktion 328
OffsetViewportOrg-Funktion 328
OffsetWindowOrg-Funktion 329
OpenClipboard-Funktion 329
OpenComm-Funktion 330
OpenFile-Funktion 331
OpenIcon-Funktion 333
OpenSound-Funktion 333
Out_Xxxx-Konstanten 50
OutputDebugString-Prozedur 333

P
PaintRgn-Funktion 334
PaletteIndex-Funktion 334
Paletten
  auswählen 360
  Einträge 570
  Größe ändern 351
  Handles 546
  Index 554
  löschen 144
  realisieren 344
  System- 244, 393
  virtuelle 565
PaletteRGB-Funktion 334
PatBlt-Funktion 335
PBool-Typ 548
PByte-Typ 548
pc_Xxxx-Konstanten 51
PeekMessage-Funktion 336f.
PHandle-Typ 548
Pie-Funktion 338
Pinsel
  erzeugen 106, 116, 121ff.
  Handles 238, 545
  löschen 144
  Stile 6
  Ursprung 188, 364
  virtuelle 563
PInteger-Typ 548

Pixel
  Farbe ermitteln 229
  setzen 383
PlayMetaFile-Funktion 339
PlayMetaFileRecord-Prozedur 339
PLongInt-Typ 548
pm_Xxxx-Konstanten 52
Polygon-Funktion 339
Polygone
  Füllmodus 229, 384
  mehrere 340
  zeichnen 339
Polygonfüllmodi-Konstanten 52
Polyline-Funktion 340
PolyPolygon-Funktion 340
PostAppMessage-Funktion 341
PostMessage-Funktion 341
PostQuitMessage-Prozedur 342
pr_JobStatus-Konstante 52
proc_Xxxx-Konstanten 52
Prozeduren
  far 563
  Instanzen 184
ps_Xxxx-Konstanten 52
PStr-Typ 548
PString-Typ 659
PtInRect-Funktion 343
PtInRegion-Funktion 343
PtrRec-Typ 659
PtVisible-Funktion 343
Puffer
  an Schnittstelle schreiben 444
  Streams 582f.
  Transferdaten 643
Punkte
  in Fenstern enthaltene 440
  in untergeordneten Fenstern 100
PWord-Typ 548

Q
Quit-Botschaft
  übermitteln 342

R
r2_Xxxx-Konstanten 53
Rahmenfenster
  Botschaftsbearbeitung 138
  Rechteck 183
  Region 183
Rasteroperationen
  ternäre 68
rc_Xxxx-Konstanten 54
ReadComm-Funktion 343
RealizePalette-Funktion 344
Rechtecke 571
  abrunden 352
  Aktualisierungsregion 251
  ausfüllen 178
  Clipping 195, 282
  Farben invertieren 285
  Gültigkeit überprüfen 438
  Koordinaten 385
  kopieren 104
  leere 289
  Punkte innerhalb 343
  sichtbare 346
  Überschneidungsbereich 283
  umgrenzende 256
  ungültig machen 283
  vereinigen 433
  vergleichen 170
  vergrößern 278
  verschieben 327
  zeichnen 345
Records
  Metafiles 568
Rectangle-Funktion 345
RectInRegion-Funktion 345
RectVisible-Funktion 346
Referenzzähler
  abfragen 224
Region-Flags 54
Regionen
  Aktualisierungs- 251
  auf Gültigkeit prüfen 438
  ausfüllen 179
  Clipping- 282
  elliptische 118
  Farben invertieren 285
  Handles 546
  kombinieren 103
  löschen 144
  Polygon- 126
  Punkte innerhalb von 343
  Rahmen zeichnen 183
  Rechtecke erzeugen aus 386
  rechteckige 128
  ungültig machen 284
  vergleichen 171
  verschieben 328
  zeichnen 334
RegisterClass-Funktion 346
RegisterClipboardFormat-Funktion 347
RegisterType-Prozedur 659

RegisterWindowMessage-Funktion 347
ReleaseCapture-Prozedur 348
ReleaseDC-Funktion 348
RemoveFontResource-Funktion 348
RemoveMenu-Funktion 349
RemoveProp-Funktion 350
ReplyMessage-Prozedur 350
ResizePalette-Funktion 351
Ressourcen
  auffinden 179
  Bitmaps 294
  freigeben 185
  Größe von 419
  laden 301
  Mauszeiger 295
  Menü- 298
  Schriften 492
  Speicher belegen für 83
  sperren 309
  Strings laden 301
  Zugriff auf 79
Ressourcentyp-Konstante 54
RestoreDC-Funktion 351
RGB-Farben 553, 572
RGB-Funktion 352
rgn_Xxxx-Konstanten 54
Rollende Fenster 624f., 627, 637
  Bereich 625
  Besitzer 624
  Einheiten 625
  Position 625f.
  Seitengröße 625
  Streams und 626
RoundRect-Funktion 352
rt_Xxxx-Konstanten 55

S

s_Xxxx-Konstanten 55
SafetyPoolSize-Variable 659
Sample-Prozedur 653
SaveDC-Funktion 353
sb_Xxxx-Konstanten 57
sbs_Xxxx-Konstanten 57
sc_Xxxx-Konstanten 59
ScaleViewportExt-Funktion 354
ScaleWindowExt-Funktion 354
Schalter 584
  auswählen 98
  Benachrichtigungscodes 5
  Konstruktor 585
  Objekte verbinden mit 585
  Ressourcen und 585
  Status 449f.
  Stile 6, 450
  Text 495
  voreingestellter 584
Schaltfelder 620
  Konstruktor 620
  markieren 99, 450
  Status 449
Schlüssel
  sortierte Kollektionen 629
Schnittstelle
  Ereignismaske 199, 370
  Fehler 197
  öffnen 330
  schließen 102
  Status 371, 555
  Steuerungsblock 199
  Übertragungsunterbrechung 101, 370
Schnittstellen
  empfangen 432
  erweiterte Funktionen 172
  neu initialisieren 371
  Puffer schreiben an 444
  Zeichen lesen 343
  Zeichen senden 432
Schnittstellenkonfigurations-Konstanten 59
Schnittstellenobjekte 641
  aktivieren 650
  automatische Erzeugung 645
  Befehlsbotschaften und 650
  darstellen 649
  Daten transferieren 649
  Destruktor 643
  einrichten 649
  entfernen und freigeben 645, 651
  Fenster-Handles 642
  Fensterklassen 647
  Flags setzen 649
  IDs 647
  Klassenbezeichnungen 647
  Konstruktor 643
  rollen 644, 651
  schließen 643, 650
  Standardbotschaftsverarbeitung 644
  Status 642
  Streams und 643
  Transfermechanismus 645f.
  untergeordnete Fenster 644
Schnittstellenschlange
  entleeren 182

Schriftartbezeichnung
  aktuelle 250
Schriftdarstellungsqualitäts-
  Konstanten 60
Schriften
  abbilden 379
  Abmessungen 250
  auflisten 164f.
  Dialogfenster und 493
  erzeugen 118f.
  Handles 546
  löschen 144
  Ressourcen 81, 348
  Schriftartbezeichnung 250
  verändern 492
  virtuelle 564
ScreenToClient-Prozedur 355
ScrollDC-Funktion 355
ScrollWindow-Prozedur 356
Seite
  EMS- 656
Seitenverhältnis 186
SelectClipRgn-Funktion 357
SelectObject-Funktion 358f.
SelectPalette-Funktion 360
SendDlgItemMessage-Funktion 360
SendMessage-Funktion 361
serielle Schnittstellen
  Escape-Konstanten 28
SetActiveWindow-Funktion 362
SetBitmapBits-Funktion 362
SetBitmapDimension-Funktion 362
SetBkColor-Funktion 363
SetBkMode-Funktion 363
SetBrushOrg-Funktion 364
SetCapture-Funktion 364
SetCaretBlinkTime-Prozedur 365
SetCaretPos-Prozedur 365
SetClassLong-Funktion 365
SetClassWord-Funktion 366
SetClipboardData-Funktion 367
SetClipboardViewer-Funktion 369
SetCommBreak-Funktion 370
SetCommEventMask-Funktion 370
SetCommState-Funktion 371
SetCursor-Funktion 372
SetCursorPos-Prozedur 372
SetDIBits-Funktion 373
SetDIBitsToDevice-Funktion 374
SetDlgItemInt-Prozedur 375
SetDlgItemText-Prozedur 376
SetDoubleClickTime-Prozedur 376
SetEnvironment-Funktion 376
SetErrorMode-Funktion 377
SetFocus-Funktion 378
SetHandleCount-Funktion 378
SetKeyboardState-Prozedur 378
SetMapMode-Funktion 379
SetMapperFlags-Funktion 380
SetMenu-Funktion 380
SetMenuItemBitmaps-Funktion 381
SetMessageQueue-Funktion 381
SetMetaFileBits-Funktion 382
SetPaletteEntries-Funktion 382
SetParent-Funktion 383
SetPixel-Funktion 383
SetPolyFillMode-Funktion 384
SetProp-Funktion 384
SetRect-Prozedur 385
SetRectEmpty-Prozedur 386
SetRectRgn-Prozedur 386
SetResourceHandler-Funktion 386
SetROP2-Funktion 387
SetScrollPos-Funktion 388
SetScrollRange-Prozedur 389
SetSoundNoise-Funktion 390
SetStretchBltMode-Funktion 390
SetSwapAreaSize-Funktion 391
SetSysColors-Prozedur 392
SetSysModalWindow-Funktion 393
SetSystemPaletteUse-Funktion 393
SetTextAlign-Funktion 395
SetTextCharacterExtra-Funktion 396
SetTextColor-Funktion 396
SetTextJustification-Funktion 397
SetTimer-Funktion 398
SetViewportExt-Funktion 399
SetViewportOrg-Funktion 400
SetVoiceAccent-Funktion 401
SetVoiceEnvelope-Funktion 402
SetVoiceNote-Funktion 403
SetVoiceQueueSize-Funktion 404
SetVoiceSound-Funktion 404f.
SetVoiceThreshold-Funktion 405
SetWindowExt-Funktion 405
SetWindowLong-Funktion 406
SetWindowOrg-Funktion 407
SetWindowPos-Prozedur 408
SetWindowsHook-Funktion 409
SetWindowText-Prozedur 415
SetWindowWord-Funktion 415
ShowCaret-Prozedur 416
ShowCursor-Funktion 416
ShowOwnedPopups-Prozedur 417

ShowScrollBar-Prozedur 417
ShowWindow-Funktion 418
Sicherheitsbereich 658
  Größe 659
SizeofResource-Funktion 419
SizeXxxx-Konstanten 60
sm_Xxxx-Konstanten 61
sp_Xxxx-Konstanten 62
Speicher
  EMS- 656
  Erweiterungs- 292
  globaler 266f., 272f.
  lokaler 302, 305ff.
  reservieren 658
  Segmente 309, 434
  verkleinern 481
ss_Xxxx-Konstanten 62f.
Standardbotschaftsverarbeitung
  Dialogfenster 598
  Fenster 637, 639
  MDI 618
  Schnittstellenobjekte 644
StartSound-Funktion 419
Statische Textfelder 630f.
  Konstruktor 631
  Objekte verbinden mit 631
  Ressourcen und 631
  Streams und 631
  Textlänge 630
Steuerelemente 595
  ausgewählte Schalter 288
  benutzerdefinierte 490
  Botschaften senden an 360
  Botschaftsverarbeitung 596
  gruppierte 225
  Handle 208
  IDs 208
  Konstruktor 596
  Objekte zuordnen 596
  Ressourcen und 596
  Tabulatorstopps 226
  Text 209, 258
  zeichnen 596
Stifte
  erzeugen 125
  Handles 546
  löschen 144
  Stil 52
  virtuelle 565
Stile
  Aktionsschalter 450
  Klasse 574
  Kombinationsfenster 594
Stings
  Zeichensätze konvertieren 87
StopSound-Funktion 419
StrDispose-Prozedur 659
StreamError-Variable 661
Streams 632ff.
  DOS 601
  EMS 608
  entleeren 633
  Fehler 633
  Fehlerbedingung 635
  Fehlercodes 632, 660
  gepufferte 582f.
  Größe von 634
  lesen aus 634
  Position 634
  Registrierung 659
  schreiben in 635
  Status 632
  verkürzen 635
Streckungsmodus
  aktueller 239
  festlegen 390
StretchBlt-Funktion 420
StretchBlt-Modi 64
StretchDIBits-Funktion 421
Strings
  Abmessungen 244, 249
  an Debugger senden 333
  dynamische 659
  freigeben 659
  Handles 546
  Kollektionen von 636
  konkatenieren 313
  kopieren 314
  laden 301
  Länge 315
  reservieren 659
  schreiben 175
  Streams und 636
  vergleichen 314
  Zeichensätze konvertieren 86f.
  Zeiger 548
StrNew-Funktion 660
stXXXX-Konstanten 660
sw_Xxxx-Konstanten 65
SwapMouseButton-Funktion 423
SwapRecording-Prozedur 424
SwitchStackBack-Prozedur 424
SwitchStackTo-Prozedur 424
swp_Xxxx-Konstanten 66

Symbole
  andordnen 616
  anordnen 615f.
  entfernen 145
  erzeugen 122
  Fenster beschränken auf 103
  Handles 546
  Hintergrund 497
  Klassen 574
  laden 297
  maximieren 333
  neu zeichnen 497
  zeichnen 157
SyncAllVoices-Funktion 425
syspal_Xxxx-Konstanten 66
System-Warteschlange
  überprüfen 213
Systemfarben festlegen 392
Systemmaße abrufen 242
systemmodales Fenster
  aktuelles 240
  auswählen 393
Systempalette
  zulässiger Gebrauch der 393
Systemregister
  Werte 96
Systemverzeichnis
  Name 240

T
ta_Xxxx-Konstanten 66
TabbedTextOut-Funktion 426
Tabulatorpositionen
  Editierfelder und 465
  in Listen 475
Tasks
  aktuelles Handle 200
  Anzahl von 227
  unterbrechen 448
Tastatur
  Code-Seite ermitteln 214
  Status 214, 378
  Typ ermitteln 215
Tastatur-Handles
  Anwendungen 582
Tastatur-Schnittstelle 645
Tasten
  gedrückte 478f., 498
  losgelassene 499
  Namen ermitteln 216
  Standard- 498f.
  Status ermitteln 217
Tastendrücke 478f.
  tote Taste 487
TAtom-Typ 549
TBitmap-Typ 549
TBitmapCoreHeader-Typ 549
TBitmapCoreInfo-Typ 550
TBitmapFileHeader-Typ 551
TBitmapInfo-Typ 551
TBitmapInfoHeader-Typ 552
TByteArray-Typ 661
tc_Xxxx-Konstanten 67
TCatchBuf-Typ 553
TClientCreateStruct-Typ 553
TColorRef-Typ 553
TCompareItemStruct-Typ 554
TComStat-Typ 555
TCreateStruct-Typ 555
TDCB-Typ 556
TDDEAck-Typ 557
TDDEAdvise-Typ 558
TDDEData-Typ 558
TDDEPoke-Typ 559
TDeleteItemStruct-Typ 559
TDevMode-Typ 560f.
TDialogAttr-Typ 661
TDrawItemStruct-Typ 562
Text
  Abmessungen 244
  Ausrichtung 397
  Ausrichtungsflags 247
  Farbe 249, 396
  Fenster 258
  formatieren 483
  kopieren 482
  Länge 495
  löschen 480
  Maße 572f.
  zeichnen 427
  zusätzlicher Leerraum 248
TextOut-Funktion 427
tf_ForceDrive-Konstante 68
tf_XXXX-Konstanten 661
TFarProcTyp 563
TGlobalHandle-Typ 563
THandle-Typ 563
THandleTable-Typ 563
Throw-Prozedur 427
Timer
  setzen 398
Timer-Ereignis 290
TItemList-Typ 662
TLocalHandle-Typ 563

TLogBrush-Typ 563
TLogFont-Typ 564
TLogPalette-Typ 565
TLogPen-Typ 565
TMDICreateStruct-Typ 566
TMeasureItemStruct-Typ 566
TMenuItemTemplateHeader-Typ 567
TMessage-Typ 662
TMetaFilePict-Typ 567
TMetaHeader-Typ 568
TMetaRecord-Typ 568
TMsg-Typ 568
TMultiKeyHelp-Typ 569
TMultiSelRec-Typ 662
ToAscii-Funktion 428
TOFStruct-Typ 569
Tortendiagramme
  zeichnen 338
TPaletteEntry-Typ 570
TPattern-Typ 571
TPauntStruct-Typ 570
TPoint-Typ 571
TrackPopupMenu-Funktion 429
Transfermechanismus 645
Transferpuffer 643
TranslateAccelerator-Funktion 429
TranslateMDISysAccel-Funktion 431
TranslateMessage-Funktion 431
TransmitCommChar-Funktion 432
TRect-Typ 571
TRGBQuad-Typ 572
TRGBTriple-Typ 572
TStreamRec-Typ 663
TTextMetric-Typ 572f.
TWindowAttr-Typ 664
TWndClass-Typ 574
TWordArray-Typ 664

U
Übergeordnete Fenster 642
Umgebung
  Systemanschluß 211
UngetCommChar-Funktion 432
UnhookWindowsHook-Funktion 433
UnionRect-Funktion 433
UnlockData-Funktion 434
UnlockResource-Funktion 434
UnlockSegment-Funktion 434
UnrealizeObject-Funktion 435
UnregisterClass-Funktion 435
Untergeordnete Fenster 642, 644
  aktivieren 479
  auflisten 163
  enthaltene Punkte 100
  erzeugen 618
  MDI 140f., 505
  nächstes 648
  nebeneinander anordnen 615
  Streams und 648f.
  überlappen 615
  vorhergehendes 648
UpdateColors-Funktion 436
UpdateWindow-Prozedur 436
Ursprung
  Grafikfenster 253

V
ValidateCodeSegments-Prozedur 437
ValidateFreeSpaces-Funktion 437
ValidateRect-Prozedur 438
ValidateRgn-Prozedur 438
Version
  von Windows 253
Verzeichnis
  Windows- 257
Verzeichnislisten
  in Kombinationsfenstern 152f.
Verzeichnisse
  Namen in Listen 151
Vielecke
  siehe Polygone
virtuelle Tasten 317
  Status 186
  Zeichen konvertieren in 439
vk_Xxxx-Konstanten 69, 71
VkKeyScan-Funktion 439

W
WaitMessage-Prozedur 439
WaitSoundState-Funktion 440
Warnton 318
wb_XXXX-Konstanten 664
wep_Xxxx-Konstanten 73
wf_Xxxx-Konstanten 73
wh_Xxxx-Konstanten 74
WindowFromPoint-Funktion 440
Windows
  Version 253
  Verzeichnis 257
WinExec-Funktion 441
WinHelp-Funktion 442f.
wm_Activate-Botschaft 476
wm_ActivateAPP-Botschaft 476
wm_AskCBFormatName-Botschaft 476

wm_CancelMode-Botschaft 477
wm_ChangeCBChain-Botschaft 477
wm_Char-Botschaft 478
wm_CharToItem-Botschaft 479
wm_ChildActivate-Botschaft 479
wm_Clear-Botschaft 480
wm_Close-Botschaft 480
wm_Command-Botschaft 480
wm_Compacting-Botschaft 481
wm_CompareItem-Botschaft 481
wm_Copy-Botschaft 482
wm_Create-Botschaft 482
wm_CtlColor-Botschaft 482
wm_Cut-Botschaft 483
wm_dde_Ack-Botschaft 483
wm_dde_Advise-Botschaft 484
wm_dde_Data-Botschaft 484
wm_dde_Execute-Botschaft 485
wm_dde_Initiate-Botschaft 485
wm_dde_Poke-Botschaft 486
wm_dde_Request-Botschaft 486
wm_dde_Terminate-Botschaft 486
wm_dde_Unadvise-Botschaft 487
wm_DeadChar-Botschaft 487
wm_DeleteItem-Botschaft 488
wm_Destroy-Botschaft 488
wm_DestroyClipboard-Botschaft 489
wm_DevModeChange-Botschaft 489
wm_DrawClipboard-Botschaft 489
wm_DrawItem-Botschaft 490
wm_Enable-Botschaft 490
wm_EndSession-Botschaft 490
wm_EnterIdle-Botschaft 491
wm_EraseBkgnd-Botschaft 491
wm_FontChange-Botschaft 492
wm_GetDlgCode-Botschaft 492
wm_GetFont-Botschaft 493
wm_GetMinMaxInfo-Botschaft 493
wm_GetText-Botschaft 494
wm_GetTextLength-Botschaft 495
wm_HScroll-Botschaft 495
wm_HScrollClipboard-Botschaft 496
wm_IconEraseBkgnd-Botschaft 497
wm_InitDialog-Botschaft 497
wm_InitMenu-Botschaft 497
wm_InitMenuPopup-Botschaft 498
wm_KeyDown-Botschaft 498
wm_KeyUp-Botschaft 499
wm_KillFocus-Botschaft 500
wm_LButtonDblClk-Botschaft 501
wm_LButtonDown-Botschaft 501
wm_LButtonUp-Botschaft 502
wm_MButtonDblClk-Botschaft 503
wm_MButtonDown-Botschaft 504
wm_MButtonUp-Botschaft 504
wm_MDIActivate-Botschaft 505
wm_MDICascade-Botschaft 506
wm_MDICreate-Botschaft 506
wm_MDIDestroy-Botschaft 507
wm_MDIGetActive-Botschaft 507
wm_MDIIconArrange-Botschaft 507
wm_MDIMaximize-Botschaft 508
wm_MDINext-Botschaft 508
wm_MDIRestore-Botschaft 508
wm_MDISetMenu-Botschaft 509
wm_MDITile-Botschaft 509
wm_MeasureItem-Botschaft 509
wm_MenuChar-Botschaft 510
wm_MenuSelect-Botschaft 511
wm_MouseActivate-Botschaft 511
wm_MouseMove-Botschaft 512
wm_Move-Botschaft 513
wm_NCActivate-Botschaft 513
wm_NCCalcSize-Botschaft 513
wm_NCCreate-Botschaft 514
wm_NCDestroy-Botschaft 514
wm_NCHitTest-Botschaft 514f.
wm_NCLButtonDblClk-Botschaft 516
wm_NCLButtonDown-Botschaft 516
wm_NCLButtonUp-Botschaft 517
wm_NCMButtonDblClk-Botschaft 517
wm_NCMButtonDown-Botschaft 518
wm_NCMButtonUp-Botschaft 519
wm_NCMouseMove-Botschaft 519
wm_NCPaint-Botschaft 520
wm_NCRButtonDblClk-Botschaft 520
wm_NCRButtonDown-Botschaft 521
wm_NCRButtonUp-Botschaft 522
wm_NextDlgCtl-Botschaft 522
wm_Paint-Botschaft 523
wm_PaintClipboard-Botschaft 523
wm_PaintIcon-Botschaft 524
wm_PaletteChanged-Botschaft 524
wm_ParentNotify-Botschaft 524
wm_Paste-Botschaft 525
wm_QueryDragIcon-Botschaft 526
wm_QueryEndSession-Botschaft 526
wm_QueryNewPalette-Botschaft 526
wm_QueryOpen-Botschaft 527
wm_Quit-Botschaft 527
wm_RButtonDblClk-Botschaft 527
wm_RButtonDown-Botschaft 528
wm_RButtonUp-Botschaft 529
wm_RenderAllFormats-Botschaft 530

wm_RenderFormat-Botschaft 530
wm_SetCursor-Botschaft 530
wm_SetFocus-Botschaft 531
wm_SetFont-Botschaft 531
wm_SetRedraw-Botschaft 532
wm_SetText-Botschaft 532
wm_ShowWindow-Botschaft 533
wm_Size-Botschaft 533
wm_SizeClipboard-Botschaft 534
wm_SpoolerStatus-Botschaft 535
wm_SysChar-Botschaft 535
wm_SysColorChange-Botschaft 536
wm_SysCommand-Botschaft 536f.
wm_SysDeadChar-Botschaft 538
wm_SysKeyDown-Botschaft 538
wm_SysKeyUp-Botschaft 539
wm_TimeChange-Botschaft 541
wm_Timer-Botschaft 541
wm_Undo-Botschaft 541
wm_VKeyToItem-Botschaft 541
wm_VScroll-Botschaft 542
wm_VScrollClipboard-Botschaft 543
wm_WinIniChange-Botschaft 543
wm_XXXX-Konstanten 665
Word Zeiger 548
WordRec-Typ 665
WriteComm-Funktion 444
WritePrivateProfileString-Funktion 444
WriteProfileString-Funktion 446
ws_ex_Xxxx-Konstanten 76
ws_Xxxx-Konstanten 74f.
wvsprintf-Funktion 447

Y
Yield-Funktion 448

Z
Zeichen
  Abstand 396
  alphabetische 286
  alphanumerische 286
  Botschaften 431
  empfangen 432
  großgeschriebene 287
  in Puffer formatieren 447
  kleingeschriebene 286
  konvertieren 326f., 439
  übertragen 432
  von Schnittstelle einlesen 343
Zeichenabstands-Konstanten 77
Zeichenbreite 190
Zeichenmodus
  aktueller 236
  festlegen 387
Zeichenobjekte
  vordefinierte 72
Zeichensatz-Konstanten 77
Zeichensätze
  ändern 86f.
Zeichenwerkzeuge
  erzeugen 554
  zurücksetzen 435
Zeichnen
  beenden 162
  Fenster 91, 639
Zeiger
  boolesche 548
  Bytes 548
  Handles 547f.
  Integer 548
  long-Integer 548
  Strings 548
  Word 548
Zeit
  seit Systemstart 200
Zusätliche Bytes
  Klasse 574
Zwischenablage
  Besitzer 195
  Daten abrufen 193
  Daten-Handle 367
  Format 476
  Kette 96, 369
  leeren 159, 489
  neu zeichnen 496
  öffnen 329
  rollen 496
  schließen 102
  Viewer 195
  Viewer-Kette 489

WINDOWS-
REFERENZ-
HANDBUCH

TURBO PASCAL®

für Windows

BORLAND

BORLAND GMBH, LINDWURMSTRASSE 88, POSTFACH 15 03 40, D-8000 MÜNCHEN 2, TELEFON 0 89/7 20 10-0, TELEFAX 0 89/77 93 39, TELEX 5 212 637 mcm d,
CORPORATE HEADQUARTERS: 1800 GREEN HILLS ROAD, P.O. BOX 660001, SCOTTS VALLEY, CA 95067-0001. (408) 438-530. OFFICES IN:
AUSTRALIA, DENMARK, FRANCE, GERMANY, ITALY, JAPAN, NEW ZEALAND, SINGAPORE, SWEDEN AND THE UNITED KINDOM * ARTIKEL-NR. 112304